Dynamic Token Bucket (DTB): A Fair Bandwidth Allocation Algorithm for High-Speed Networks [1]

Jayakrishna Kidambi†, Dipak Ghosal‡, and Biswanath Mukherjee‡
†Department of Electrical and Computer Engineering, University of California, Davis
‡Department of Computer Science, University of California, Davis
jkidambi@ece.ucdavis.edu, ghosal@cs.ucdavis.edu, mukherje@cs.ucdavis.edu

Abstract

Fair allocation of available bandwidth to competing flows is a simple form of quality of service (QoS) that can be provided to customers in packet-switched networks. A number of packet-scheduling and buffer-management techniques have been proposed in the literature to achieve this goal efficiently. However, the complexity of the existing algorithms prevents a high-speed implementation with the current state of router technology. We propose a computationally simple mechanism based on token-bucket policing to achieve almost equal bandwidth allocation for a set of competing flows. The proposed method adjusts the token-bucket threshold dynamically and measures the instantaneous arrival rate of flows. It uses this information to decide whether or not to admit a packet arriving at the network edge. With minor modifications, our framework can be used in the Internet and Frame Relay based virtual private networks (VPNs). We present a detailed simulation study that evaluates the performance of our algorithm. The simulation results indicate that DTB is fair, efficient, and robust.

Keywords: High-speed packet-switched networks, flows, bandwidth allocation, fairness, dynamic token bucket.

I. INTRODUCTION

Current public data networks are typically based on the best-effort model, in which network routers maintain the first-in-first-out (FIFO) queuing mechanism. This allows little control over bandwidth distribution to different end-to-end applications. In a typical FIFO scheduling scenario, bandwidth allocation is proportional to the arrival rate of a flow. In our work, a flow can be considered to be a TCP/IP flow or a switched virtual circuit in the case of connection-oriented networks such as Asynchronous Transfer Mode (ATM) or Frame Relay. We consider a router mechanism to be fair if all competing flows receive an equal share of the link bandwidth. In public networks, since service cannot be prioritized, this definition of fairness ensures that misbehaving flows do not monopolize link capacity.

FIFO scheduling performs poorly with respect to fair bandwidth allocation. Protocols such as TCP which use congestion control are also not immune to the inherent unfairness in FIFO scheduling. Previous work [1, 2, 3] has shown that bandwidth-management techniques deployed in network routers can help attain some degree of fairness. However, intelligence in routers typically means maintaining per-flow information and performing per-flow computations. This sort of additional data processing requirements raise important issues such as scalability and routing efficiency. Complex algorithms could achieve the desirable degree of fairness while slowing down the routing operation. Section II describes some of these algorithms proposed previously in the literature.

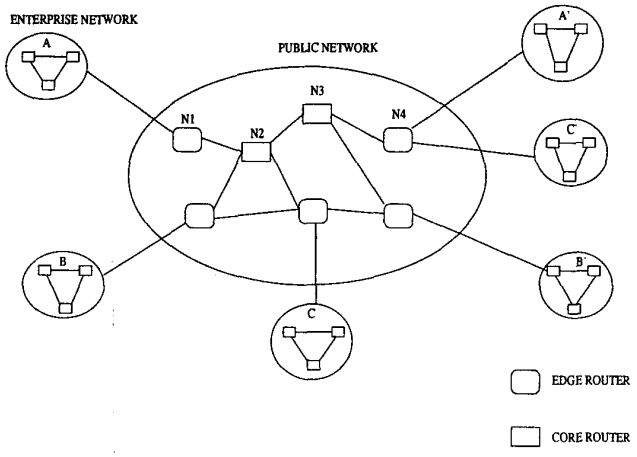

Figure 1: Network architecure considered in this study.

We propose a new bandwidth management algorithm based on token-bucket policing, called the dynamic token-bucket (DTB) algorithm. Section II describes previous work, while Section III describes the DTB algorithm in detail. Section IV provides a simulation study of DTB. In order to compare our results with a benchmark, we use the simulation scenarios described in [4] for core stateless fair queuing (CSFQ). All our simulations were performed using the Network Simulator (ns) software [5]. Section V outlines network architectures where our framework can be effectively utilized. Section VI discusses the limitations of our approach and suggests future work.

II. PREVIOUS WORK

Fair bandwidth allocation focuses on developing router mechanisms to protect users from traffic that is not responsive to congestion. There are two distinct techniques - scheduling and buffer management. Scheduling algorithms maintain multiple queues and per-flow information and seek to allocate equal

[1]This work was supported by SBC Technologies and the UC MICRO Program.

service time to each flow. Buffer-management techniques, on the other hand, use a scheme in which the router drops packets probabilistically even when the buffer is not full, by detecting congestion early. In this section, we briefly describe some typical buffer-management and packet-scheduling algorithms.

Random early detection (RED) [7] is a popular buffer-mangement strategy which is implemented in some networks, notably Frame Relay. RED routers compute the average queue size continuously. When the average queue size exceeds a preset threshold, the router drops or marks each packet with a certain probability which depends on the instantaneous queue size. RED does not maintain per-flow information. However, a variant of RED, called flow random early drop (FRED) [8] maintains statistics of buffer occupancy to improve upon the unfairness of RED in the presence of misbehaving flows.

Scheduling mechanisms such as weighted fair queueing (WFQ) [9] and generalized processor sharing (GPS) [10] are based on the principles of a fluid-flow model. In this model, flows are viewed as bit streams rather than as packets and it is assumed (as an abstract model) that multiple queues can be serviced at the same time. A fair-queueing scheduler achieves ideal fairness by offering the same service (normalized to the ideal rate) to all back-logged flows. However, in real systems, multiple queues cannot be serviced simultaneously and bit streams are quantized into packets. A number of algorithms which take this factor into account have been proposed. For instance, deficit round robin (DRR) [11] approximates WFQ by dropping packets from the longest queue when the buffer is filled to capacity.

Buffer-management techniques such as RED and FRED are simple but not totally effective for fair bandwidth allocation. Scheduling algorithms such as DRR come close to the ideal GPS model, but they are complex. CSFQ, proposed in [4], seeks to strike a balance between complexity and efficiency. In CSFQ, a distinction is made between core routers and edge routers, as illustrated in Figure 1. In CSFQ, edge routers maintain per-flow state information and the scheduling algorithm is comparable in complexity to FRED. Core routers maintain no per-flow information and the algorithm used at core routers is comparable in complexity to RED. CSFQ estimates the arrival rate of a flow by exponentially averaging the measured arrival rates. From this information, it computes the instantaneous fair rate for the link and it labels each packet with the fair rate or the estimated arrival rate, whichever is smaller. This information is used by the downstream routers. Though it dispenses with per-flow queueing, CSFQ introduces the overhead of packet relabeling. In order to compare the results of our work, the simulation scenarios proposed in [4] are taken as a benchmark.

III. Dynamic Token-Bucket (DTB) Algorithm

The token-bucket mechanism [12] is a simple traffic-shaping approach that permits burstiness, but bounds it. Traffic shaping is enforced by maintaining a counter to count tokens and a timer to determine when to add new tokens to the counter. When a packet of length l (bytes) arrives at the buffer and there are at least l tokens in the counter, the packet is sent and l tokens are removed from the counter. If there are less than l tokens in the token bucket, the packet must wait for tokens to drip into the bucket, until there are at least l tokens, at which time it is sent. The token bucket is replenished at an effective rate of RT_c tokens every T_c seconds. The advantages of the token bucket is its simplicity and its ability to accommodate limited burstiness which is typical of data traffic. There is a trade-off in the selection of T_c. A higher value of T_c allows better burst performance, while a small value of T_c helps prevent flows from sending a large burst in a very small interval of time. A timer that resets every T_c seconds triggers re-initialization of the token bucket.

DTB is based on the token-bucket algorithm with some modifications. In DTB, each active flow is allocated a token bucket. The capacity of the token bucket is equal to the instantaneous fair rate, F(t). The fair rate depends on the number of active flows N(t) and the link capacity C as follows:

$$F(t) = \frac{C}{N(t)} \tag{1}$$

For each flow, indexed by f_i, DTB maintains the following variables in a router. (1) $arrivals_i$, the number of bytes of f_i arriving at the buffer since the last time the token buckets were replenished, (2) $admitted_i$, the number of bytes of f_i admitted into the buffer since the last time the buckets were filled, and (3) $active_i$, a boolean variable that indicates whether the flow is active or not. A flow is declared to be active when it sends its first packet after the timer is reset. An active flow is declared inactive if it sends no packets during the interval between two successive timer resettings.

Apart from the above per-flow state variables, DTB also maintains the global variables F(t) and N(t). We define MAXFLOWS to be a constant equal to the maximum number of flows supported on any link. We set $active_i$ to zero for all values of i. When the first packet from flow f_i arrives at the link, $active_i$ is set to 1 to indicate that the flow is active. N(t), which is initially reset, is incremented by 1 and the corresponding value for the fair rate F(t) is computed and stored. At this time, $arrivals_i$ and $admitted_i$ are set to zero, effectively initializing the token bucket for f_i. After the initialization, whenever a packet belonging to f_i is admitted into the buffer, $admitted_i$ is incremented by the length of the packet, l_i. However, the packet is admitted only if, at time t,

$$admitted_i + l_i \leq F(t) \times T_c \tag{2}$$

If Eqn. (2) is not true, the packet is dropped immediately, unlike the basic token-bucket approach in which it is saved in a buffer until there are l_i tokens in the bucket. This is done to avoid per-flow buffering overhead in favor of a simple FIFO scheduling. Just as in the basic token-bucket implementation, a timer goes off every T_c seconds, at which time $admitted_i$ and $arrivals_i$ are cleared for all active flows.

We now explain the significance of the state variable $arrivals_i$. We have observed in our simulations (see Section 4) that Eqn. (2) by itself will not achieve fairness. When a number of flows send packets at different rates, it has been observed that the flows with low transmission rates achieve very low throughputs (normalized to the fair rate). To handle this problem, we define the variable $arrivals_i$ for each active flow. It is initialized when the first packet of a previously inactive flow arrives at the buffer and is incremented by the length of each arriving packet of the corresponding flow. From this measurement, the arrival rate of the flow is estimated. When a packet belonging to flow f_i arrives at the buffer and there are enough tokens and buffer space available, a probabilistic decision is made on whether or not the packet should be admitted. The packet is dropped if (at time t) the buffer level is above a pre-defined threshold C_{thresh} and

$$\mathbf{rand}(0, 1] > \frac{F(t)}{R_i(t)} \tag{3}$$

where **rand**(0, 1] generates a uniform random variable between 0 and 1, F(t) is the instantaneous fair rate defined by Eqn. (1), and $R_i(t)$ is the estimated arrival rate defined by:

$$R_i(t) = \frac{arrivals(\hat{t})}{\hat{t}} \tag{4}$$

where $\hat{t}$ is the value of the timer since the previous time it expired. Eqn. (4) represents a simple scheme. Better averaging techniques such as exponential weighted mean as used in RED can also be used . The value of C_{thresh} is selected so as to set aside buffer space for weak flows by dropping packets of misbehaving flows (flows transmitting at well above the fair rate) probabilistically. Observe that Eqn. (3) cannot be true if the arrival rate of a flow is less than or equal to the fair rate. The greater the arrival rate, the greater is the drop probability. Hence, weaker flows are effectively shielded against misbehaving flows.

A. Distribution of Excess Bandwidth

Like most bandwidth-allocation algorithms reported in literature, DTB cannot achieve perfect fairness when it comes to dividing unused bandwidth among competing flows. For example, assume that a flow is transmitting at a rate significantly lower than the fair rate. The bandwidth not used by this flow should ideally be divided equally among the other flows. Using DTB as described in Algorithm 1, the link utilization would fall since the bandwidth left over by the weak flow cannot be used by the other flows. However, a minor change to the algorithm can improve upon this easily. Instead of dropping all packets of a flow whose token bucket is empty, we admit packets into the buffer selectively, allowing flows to send at rates above the fair rate if there is some capacity left unused by the other flows. These packets could be marked discard eligible (DE), so that downstream routers can identify and discard these packets in the event of congestion. For this purpose, we can define buffer thresholds that identify various degrees of congestion, similar to a typical Frame-relay implementation. For instance, if the buffer is more than 80% full, all DE packets are discarded and all admitted packets are marked DE.

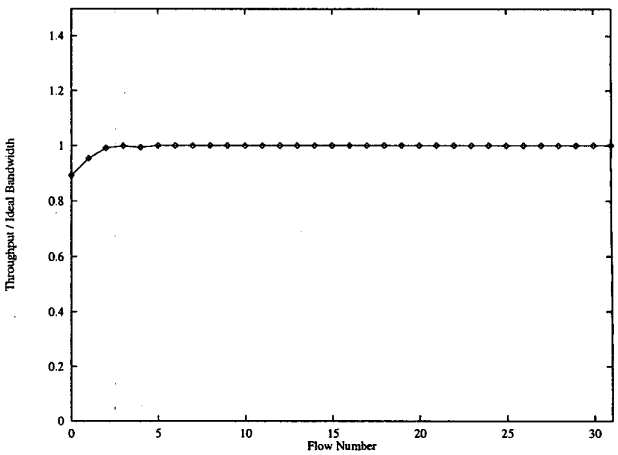

Figure 2: Normalized throughput of 32 UDP sources, indexed i, where $0 \leq i < 31$, on a 10-Mbps link with DTB.

IV. Illustrative Numerical Examples and Simulations

Our simulation experiments closely follow that of [4]. In our simulations, we use different protocol types and traffic sources. UDP sources are used to simulate misbehaving flows. TCP sources are used to represent the class of responsible flows that employ end-to-end congestion control. We also use exponential on/off and Pareto on/off sources to approximate various degrees of burstiness, which is a key characteristic of internet traffic [13].

Unless stated otherwise, we assume the following default parameters: Each link has a capacity of 10 Mbps, a propagation delay of 1 ms, and a buffer of size 64 KB. Packet length is 1 KB, including the header. All TCP flows are from continuously back-logged sources. For the DTB algorithm, C_{thresh} is half the buffer capacity, and T_c is 1 second. All throughput measurements are averaged over a 20-second interval. A detailed sensitivity analysis of the above parameters is beyond the scope of this paper.

A. UDP Flows Sharing a Single Congested Link

First, we simulate a single 10-Mbps congested link that is shared by UDP flows sending packets at different rates. The sources are indexed by UDP_i, where $0 \leq i \leq 31$. The fair rate is $\frac{10}{32} = 0.3125$ Mbps. The rate at which UDP_i sends packets is $0.3125 * (i + 1)$. Thus, UDP_0 sends at 0.3125 Mbps, UDP_1 sends at 0.6250 Mbps, UDP_2 sends at 0.9375 Mbps, etc. Figure 2 shows the throughput of the 32 flows normalized with respect to *min*(average arrival rate, fair rate). We observe that DTB allocates fair share across all sources irrespective of their arrival rates. However, there is still a slight amount of unfairness for UDP_0 and UDP_1.

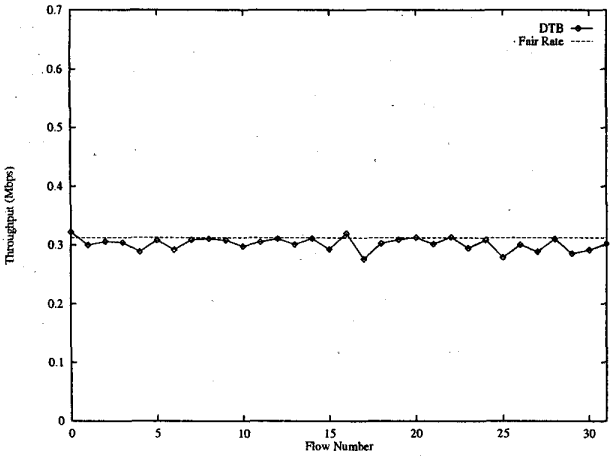

Figure 3: Throughput of one UDP source (flow 0, 10 Mbps) and 31 TCP sources sharing a 10-Mbps link.

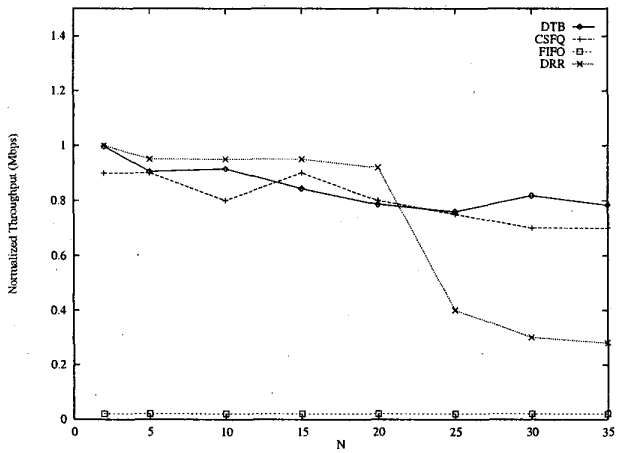

Figure 4: Normalized bandwidth of the TCP flow that competes with $N-1$ UDP flows, each of which sends at twice the fair rate.

B. Impact of an Ill-Behaved UDP Flow on a Set of TCP Flows

In this simulation, 31 TCP flows (indexed 1 to 31) compete with a UDP flow (flow 0) that sends at 10 Mbps. Figure 3 shows the average throughput of all the 32 flows, in Mbps. DTB performs very well in this case. We can see that all the flows are very close to the ideal value of fair bandwidth. The misbehaving UDP source is restricted to 0.322 Mbps. The throughput of the TCP sources ranges from 0.28 Mbps to 0.32 Mbps. We have noted (not shown in figure) that DTB performs much better than simple FIFO scheduling with respect to fair bandwidth allocation when all flows use TCP.

C. A Single TCP Flow Versus Multiple Ill-Behaved UDP Flows

In this simulation, we consider one TCP flow and $N-1$ ill-behaved UDP flows where $2 \leq N \leq 35$. We perform the same simulation for different values of N (2, 5, 10, 15, 20, 25, 30, and 35). Each UDP flow sends packets at twice the fair rate. For instance, when $N = 2$, the UDP source sends at 10 Mbps. When $N = 5$, the four UDP sources send at 4 Mbps each. Figure 4 shows the average throughput of the TCP source over a 20-second interval, using different algorithms: DTB, CSFQ, FIFO, and DRR. The throughputs of the UDP sources (not shown here), have been verified to be close to the values shown in Figure 4. The results reported in [4] show that DRR performs very well upto $N = 25$, after which it breaks down. CSFQ performs well across the range of N, though it is not as efficient as DRR for $N < 25$. FIFO's performance is predictably poor. DTB performs reasonably well in these conditions for the entire range of N. While the normalized throughput has a gradual decreasing trend as N increases, it compares favorably with CSFQ.

D. Multiple Congested Links: A Realistic Network Setting

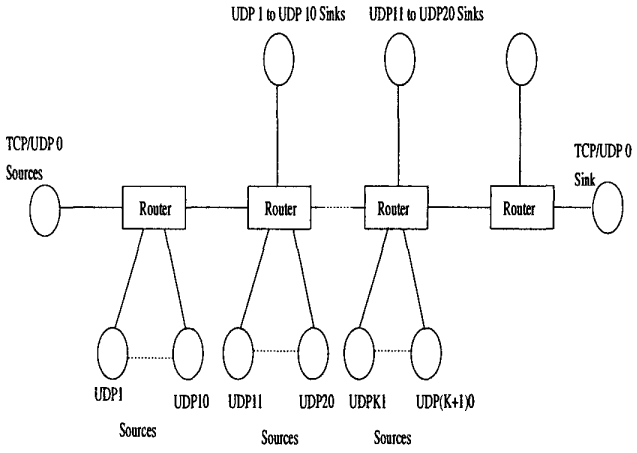

Figure 5: Network of routers.

In order to evaluate the performance of DTB in a realistic network setting, we consider the network as shown in the Figure 5. We consider a tagged flow, flow 0, which traverses M links where $1 \leq M \leq 5$. The background traffic on each link consists of 10 UDP sources, each sending at 2 Mbps. As a result, all the links which flow 0 traverses are congested. Figures 6 and 7 show the throughput of flow 0 as a function of M. DTB, CSFQ, and DRR show similar performance for the case when flow 0 is a UDP source with a rate of 0.909 Mbps, which is the fair rate. When flow 0 is from a back-logged TCP source, DTB performs better than CSFQ.

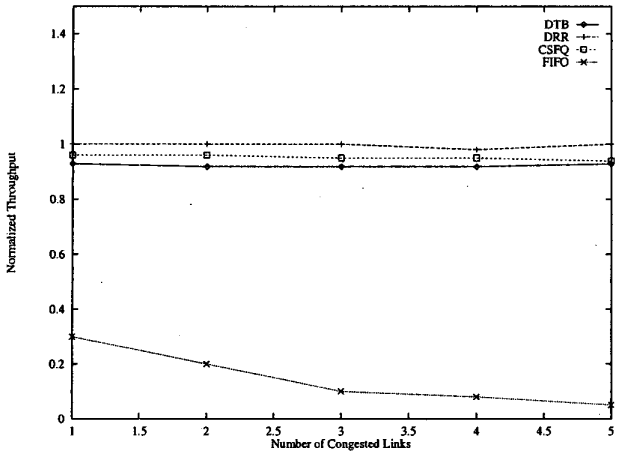

Figure 6: Normalized throughput of UDP0 as a function of the number of links M.

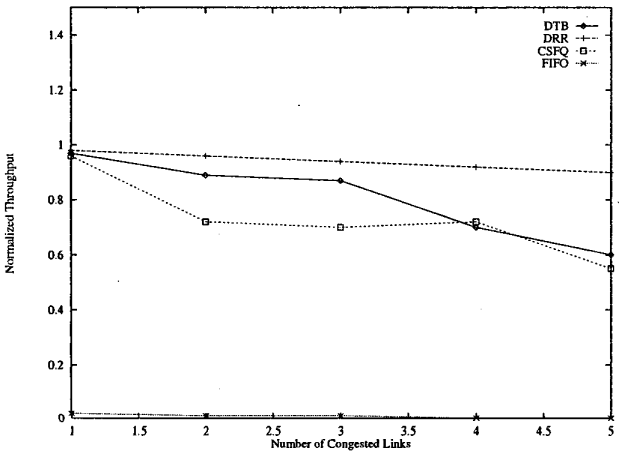

Figure 7: Normalized throughput of TCP0 as a function of the number of links M.

E. Performance in Bursty-Traffic Conditions

We also studied the performace of DTB in a bursty-traffic environment. The first experiment involves a 10-Mbps link shared by one UDP source and 19 TCP flows. The UDP source is an exponential on/off source with the on time and off time distributed with a mean of 100 ms and 1900 ms, respectively. During the on state, the source sends at a rate of 10 Mbps and is silent during the off time. Table 4.1 shows the number of packets delivered and the number of packets dropped during this interval for CSFQ, FIFO, and DTB. As expected, FIFO performs the worst, because it drops the fewest packets as it cannot recognize bursts. CSFQ and DTB prevent the 10-Mbps exponential on/off source from monopolizing the link bandwidth. In this case, DTB is better than CSFQ since it drops more packets, in accordance with the observations in [4] that an algorithm that is fair at all times will drop most of the packets during the burst.

Table 1
Number of packets of the exponential on/off source dropped and delivered at the link.

	Dropped packets	Delivered packets
FIFO	1263	4615
CSFQ	2547	3331
DTB	3942	1936

The second experiment simulates web traffic. The background traffic is generated by a 10-Mbps UDP source. Web traffic is simulated by a set 60 TCP transfers, whose interarrival times are exponentially distributed with a mean of 0.05 ms. The length of each TCP session has a Pareto distribution [13] with a mean of 20 packets and a shape parameter of 1.06 (values consistent with those in [15]). Again, we simulated this scenario for FIFO, CSFQ, and DTB. The mean and standard deviation of the transfer times of the 60 TCP transfers are shown in Table 4.2. We observe that DTB achieves a much better mean transfer time than CSFQ and FIFO. Notice that the standard deviation of the transfer time for DTB is higher than that for CSFQ. This should not be a cause for concern since the variance of the Pareto distribution is quite high and the transfer times for DTB follow the burst lengths faithfully.

Table 2
Mean and standard deviation of the transfer time of the 60 TCP sessions which simulate web traffic.

	Mean	Standard Deviation
FIFO	410	1351
CSFQ	47	51
DTB	23	123

F. Application to Practical Network Settings

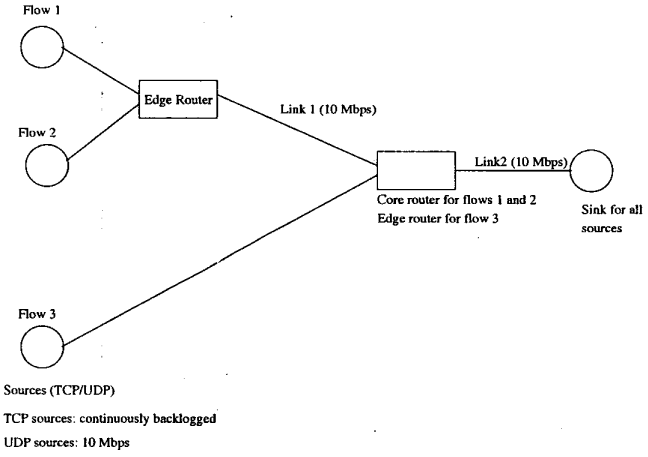

Figure 8: Network setting to test the applicability of DTB to various architectures.

We now consider a network shown in Figure 8 with three sources competing for link bandwidth on two bottleneck

links of capacity 10 Mbps each. On Link 1, only flows 1 and 2 compete for link bandwidth whereas, on Link 2, all three sources are active.

Table 4.3 shows the throughput of the flows over a 20-second interval when DTB is employed at both the edge and the core routers. DTB achieves almost perfect fairness for both TCP and UDP cases.

Table 3
Throughput of the three flows on link 2 when DTB is deployed at both the routers.

Protocol	flow 1	flow 2	flow 3
UDP	3.322	3.333	3.335
TCP	3.258	3.238	3.258

Table 4.4 shows the throughput of the three flows when only aggregates are recognized at the core. We notice that DTB works efficiently for both network scenarios. The throughputs are very close to the respective ideal values as above.

Table 4
Throughput of the three flows on link 2 when the core router identifies only traffic aggregates.

Protocol	flow 1	flow 2	flow 3
UDP	2.532	2.468	4.997
TCP	2.452	2.494	5.000

The two network settings simulated in this section correspond to two practical network architectures. The first setting in which each router maintains state information for all flows traversing the link can be deployed in the Internet to restrict all flows on a link to the fair rate. The second simulation corresponds to a Frame-Relay environment in which permanent virtual circuits (PVCs) are configured on all the links along the path for traffic flowing from one router to another.

V. Conclusion

The simulation results shown in the previous section demonstrate that DTB is an effective algorithm for fair bandwidth allocation. It is simple to implement and compares favorably with the best of fair allocation algorithms. By assigning weights to flows, we can implement a differentiated-services model in a network using DTB. In this model, unlike fair bandwidth allocation, flows can be pre-configured to receive differentiated bandwidth. In this case, each token bucket will have a different capacity or threshold, depending on the service it signed up for. When an inactive flow is activated, its token bucket is initialized to the corresponding pre-configured threshold, instead of the fair rate as in DTB.

A drawback of DTB is that we might not be able to attain perfect fairness in excess bandwidth distribution, in spite of using the discard-eligible bit as mentioned in Section III. A simple and efficient mechanism to accomplish this should be researched further.

VI. References

[1] J. Nagle, "On packet switches with infinite storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435-438, April 1987.

[2] S. Floyd and K. Fall, "Router mechanisms to support end-to-end congestion control," LBL Technical Report, February 1997.

[3] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365-386, August 1995.

[4] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high-speed networks," *Computer Communication Review* (Proceedings of ACM SIGCOMM '98, Vancouver, Canada), vol. 28, no. 4, pp. 118-130, October 1998.

[5] S. McCanne and S. Floyd, ns – Network Simulator, *http://www-mash.cs.berkeley.edu/ns/*

[6] S. Ortiz, Jr., "Virtual private networks: leveraging the Internet," *IEEE Computer*, vol. 30, no. 11, pp. 18-20, November 1997.

[7] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, August 1993.

[8] D. Lin and R. Morris, "Dynamics of Random Early Detection," *Computer Communication Review* (Proceedings of ACM SIGCOMM '97, Cannes, France), vol. 27, no. 4, pp.127-137, September 1997.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Internetworking: Research and Experience*, pp. 3-26, 1990.

[10] A. K. Parekh and R. G. Gallager, "A generalized processer sharing approach to flow control - the single router case," *Proceedings of IEEE INFOCOM '92*, Florence, Italy, vol. 2, pp. 915-924, May 1992.

[11] M. Sreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *Computer Communication Review* (Proceedings of ACM SIGCOMM '95, Cambridge, MA), vol. 25, no. 4, pp. 231-243, September 1995.

[12] D. S. Holtsinger and H. G. Perros, "Performance analysis of leaky bucket policing mechanisms," *Asia-Pacific Engineering Journal*, vol. 3, pp. 235-264, September 1993.

[13] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226-244, June 1995.

[14] W. Richard Stevens, *TCP/IP Illustrated, Volume 1*, Addison Wesley, 1994.

[15] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic evidence and possible causes," *Performance Evaluation Review* (Proceedings of ACM SIGMETRICS '96, Philadelphia, PA), vol. 24, no. 1, pp. 160-169, May 1996.