Journal of Computer Sciences and Applications, 2018, Vol. 6, No. 2, 75-81
Available online at http://pubs.sciepub.com/jcsa/6/2/4

DOI:10.12691/jcsa-6-2-4

Modeling Traffic Shaping and Traffic Policing in Packet-Switched Networks

Wlodek M. Zuberek[1,*], Dariusz Strzeciwilk[2]

[1]Department of Computer Science, Memorial University, St. John's, Canada
[2]Department of Applied Informatics, University of Life Sciences, Warsaw, Poland
*Corresponding author: wzuberek@mun.ca

Received August 26, 2018; Revised October 01, 2018; Accepted October 26, 2018

Abstract Traffic shaping is a computer network traffic management technique which delays some packets to make the traffic compliant with the desired traffic profile. Traffic policing is the process of monitoring network traffic for compliance with a traffic contract and dropping the excess traffic. Both traffic shaping and policing use two popular methods, known as leaky bucket and token bucket. The paper proposes timed Petri net models of both methods and uses these models to show the effects of traffic shaping and policing on the performance of very simple networks.

Keywords: *traffic shaping, traffic policing, packet-switched networks, leaky bucket algorithm, token bucket algorithm, timed Petri nets, performance analysis*

Cite This Article: Wlodek M. Zuberek, and Dariusz Strzeciwilk, "Modeling Traffic Shaping and Traffic Policing in Packet-Switched Networks." *Journal of Computer Sciences and Applications*, vol. 6, no. 2 (2018): 75-81. doi: 10.12691/jcsa-6-2-4.

1. Introduction

Traffic shaping is a computer network traffic management technique which delays some packets to make the traffic compliant with the desired traffic profile [1]. Traffic shaping is used to improve network latency and/or increase usable bandwidth for some classes of packets by delaying packets of other classes [2].

If a communication link is used to the point when there is a significant level of congestion, latency can rise substantially. Traffic shaping can be used to prevent this from occurring and keep latency in check. Traffic shaping provides a means to control the volume of traffic sent into a network in a specified period of time, or to control the maximum rate at which traffic is sent. This control can be accomplished in many different ways, however traffic shaping is typically achieved by delaying packets.

Traffic shaping can be (formally) regarded as a transformation of the probability distribution function of the inter-arrival times of packets to be transmitted over a communication channel. The effect of traffic shaping can be illustrated by a simple example of a single-channel queueing station with exponentially distributed service times. If the interarrival times are also exponentially distributed (i.e., the model is the popular M/M/1 queue [3,4]), the average packet waiting time, T_w, is:

$$T_w = \frac{\rho}{s(1-\rho)}$$

where s is the service rate and ρ is the traffic intensity.

If, in this model, the exponentially distributed inter-arrival times are replaced by deterministic arrivals (i.e., the model becomes D/M/1), the average waiting time is:

$$T_w = \frac{\rho}{2s(1-\rho)}$$

and, for the same values of ρ and s, it is two times smaller than in the M/M/1 case.

This simple example may suggest that the waiting times of packets transmitted over a communication network can be reduced by transforming the probability distribution function of their inter-arrival times. Traffic shaping techniques are used for this purpose. However, traffic shaping may introduce significant additional traffic delays, so its (potential) advantages can easily be lost.

Traffic shaping is often used at network edges to control traffic entering the network, but it can also be used at the nodes of the network.

Traffic shaping is similar, in many aspects, to traffic policing. The main difference is in dealing with packets which do not conform to the required traffic profile. In traffic shaping, such packets are usually delayed until they satisfy traffic requirements while in traffic policing they are simply dropped (or marked to be dropped later on).

A recent study [5] shows that a significant part of internet traffic is policed, and that more than 20% of the policed traffic is lost due to packets dropping, reducing the effective bandwidth of internet. A better traffic management scheme would use the bandwidth wasted by the lost traffic.

In the following sections, two popular traffic shaping methods, known as leaky bucket and token bucket, are discussed in greater detail and are represented as timed

Petri net models. These models are then used to show the effects of traffic shaping and traffic policing on the performance of simple examples.

Section 2 recalls basic concepts of Petri nets and timed Petri nets. Leaky bucket scheduling is discussed in Section 3 and token bucket scheduling in Section 4. Section 5 shows how traffic policing can be used to provide quality of service in priority queueing. Section 6 concludes the paper.

2. Timed Petri Nets

Petri nets are formal models of systems that exhibit parallel and concurrent activities [6,7]. In Petri nets, these activities are represented by *tokens* which can move within a (static) graph-like structure of the net. More formally, a marked (weighted inhibitor) place/transition Petri net $\mathcal{M}$ is defined as $\mathcal{M}=(\mathcal{N},m_0)$, where the structure $\mathcal{N}$ is a bipartite directed weighted graph, $\mathcal{N}=(P,T,A,w)$, with two types of vertices, a set of places P and a set of transitions T, a set of directed arcs A connecting places with transitions and transitions with places, $A\subseteq T\times P\cup P\times T$, and a weight function $w:A\to\{0,1,...\}$ which describes the multiplicity of arcs with the value 0 indicating inhibitor arcs; the initial marking function m_0 assigns nonnegative numbers of tokens to places of the net, $m_0:P\to\{0,1,...\}$. Marked nets can also be defined as $\mathcal{M}=(P,T,A,w,m_0)$.

A place is shared if it is connected to more than one transition. A shared place p is free-choice if the sets of places connected by directed arcs to all transitions sharing p are identical and the connecting arcs have the same weights. For each free-choice place, either all transitions sharing it are enabled by a marking m or all are disabled by m. It is assumed that a transition to occur in each free-choice class is chosen in a random way, and this choice is independent from other free-choice classes, so it can be described by a probability associated with a transition.

In timed nets [8], occurrence times are associated with transitions, and transition occurrences are real-time events, i.e., tokens are removed from input places at the beginning of the occurrence period, and they are deposited to the output places at the end of this period. All occurrences of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transitions cannot initiate their occurrences). If, during the occurrence period of a transition, the transition becomes enabled again, a new, independent occurrence can be initiated, which will overlap with the other occurrence(s). There is no limit on the number of simultaneous occurrences of the same transition (sometimes this is called infinite occurrence semantics). Similarly, if a transition is enabled "several times" (i.e., it remains enabled after initiating an occurrence), it may start several independent occurrences in the same time instant.

More formally, a timed Petri net is a triple, $\mathcal{T}=(\mathcal{M},c,f)$, where $\mathcal{M}$ is a marked net, c is a choice function which assigns probabilities to transitions in free-choice classes or relative frequencies of occurrences for conflicting transitions, $c:T\to[0,1]$, and f is a timing function which assigns an (average) occurrence time to each transition of the net, $f:T\to\mathbf{R}^+$, where $\mathbf{R}^+$ is the set of nonnegative real numbers.

The occurrence times of transitions can be either deterministic or stochastic (i.e., described by some probability distribution function); in the first case, the corresponding timed nets are referred to as D-timed nets [9], in the second, for the (negative) exponential distribution of firing times, the nets are called M-timed nets (Markovian nets) [10]. In both cases, the concepts of state and state transitions have been formally defined and used in the derivation of different performance characteristics of the model. In simulation applications, other distributions can also be used, for example, the uniform distribution (U-timed nets) is sometimes a convenient option. In timed Petri nets different distributions can be associated with different transitions in the same model providing flexibility that is used in simulation examples that follow.

In timed nets, the occurrence times of some transitions may be equal to zero, which means that the occurrences are instantaneous; all such transitions are called immediate (while the others are called timed). Since the immediate transitions have no tangible effects on the (timed) behavior of the model, it is convenient to `split' the set of transitions into two parts, the set of immediate and the set of timed transitions, and to first perform all occurrences of the (enabled) immediate transitions, and then (still in the same time instant), when no more immediate transitions are enabled, to start the occurrences of (enabled) timed transitions. It should be noted that such a convention effectively introduces the priority of immediate transitions over the timed ones, so the conflicts of immediate and timed transitions are not allowed in timed nets. Detailed characterization of the behavior or timed nets with immediate and timed transitions is given in [8].

3. Leaky Bucket Scheme

The leaky bucket algorithm is used in packet-switched networks to check that data transmissions (in the form of packets) conform to defined limits on bandwidth and burstiness. It can also be used as a scheduling algorithm to determine the timing of transmissions that will comply with the limits set for the bandwidth and burstiness [11].

The leaky bucket scheme can be regarded as a simple finite capacity queueing station [12], as shown in Figure 1, with the capacity of the queue representing the capacity of the bucket, and with (deterministic) service times corresponding to the "leaking time" (i.e., time of forming a single drop).

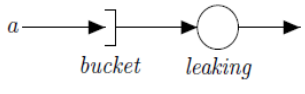

Figure 1. Leaky bucket model

The scheme limits the rate of outgoing packets by the "leaking rate" and removes the burstiness from the arriving traffic - the outgoing packets are spaced by at least the "leaking time". This shaping of the traffic is performed at the cost of a delay introduced by the leaky bucket, and this delay is a function of the "leaking rate", the arrival rate of packets, a, as well as the interarrival times of packets.

A Petri net model of leaky bucket is shown in Figure 2, in which places p_x and p_y represent the entry and the exit of the scheme, place p_b models the bucket, transition t_s with place p_s model the leaking process (one drop at a time); t_s has a deterministic occurrence time T_s associated with it. Place p_c with its initial marking K determines the capacity of the bucket, and transitions t_a and t_d either enter an incoming packet into the bucket or drop the packet if the bucket is full, i.e., if place p_c is unmarked.

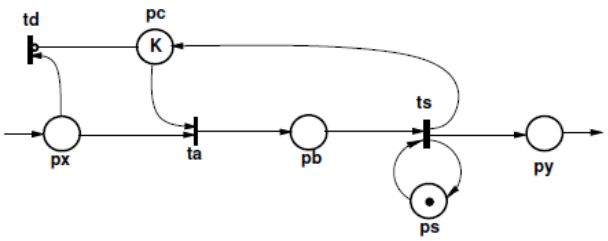

Figure 2. Petri net model of the leaky bucket scheme

Initially, place p_c is marked, so the incoming packets are forwarded to the bucket p_b and the leaking begins. After some time, if the bucket is full (and p_c is empty), the incoming packets cannot be stored in the bucket, so the occurrences of t_d remove the packets from the model.

The effects of leaky bucket scheduling is shown by comparing the performance of a simple queueing system with leaky bucket with the same system without the leaky bucket scheme. In such comparisons it is assumed that the bucket capacity is infinite, so no packets are dropped. Leaky bucket with infinite capacity can be simplified as shown in Figure 3.

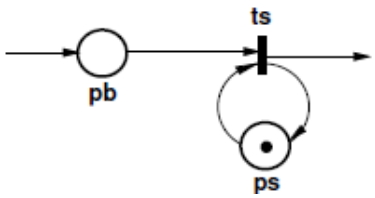

Figure 3. Petri net model of leaky bucket with infinite capacity

A simple queueing system, composed of bursty source and three consecutive service stations, as shown in Figure 4, is used as an illustration of how the leaky bucket scheme can affect the performance of a queueing system. All service times in Figure 4 are uniformly distributed between 0.5 and 1.5 time units.

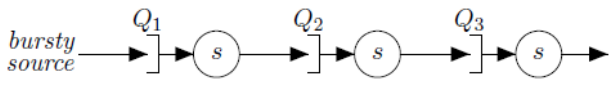

Figure 4. Queueing example

The pattern of the bursty source is 5 arrivals with 0.1 time units interarrival times followed by a single arrival after 6.1 time units (so the arrival rate is equal to 1.1). A more general Petri net model of bursty arrivals is shown in Figure 5 (for this example, $K_1 = 5$ and $K_2 = 1$).

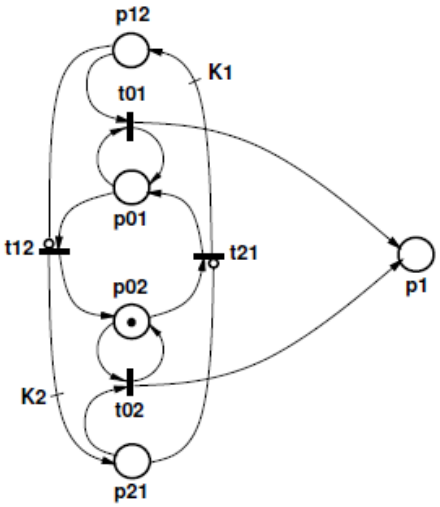

Figure 5. Petri net model of a bursty source $K_1 * T_1 + K_2 * T_2$

Timed transitions t_{01} and t_{02} generate the first and the second part of the bursty traffic, respectively. Places p_{12} and p_{21} are the counters for section 1 and 2, respectively. If place p_{12} is marked, each occurrence of transition t_{01} reduces the contents of p_{12} by one. When p_{12} becomes unmarked, the token from p_{01} is moved to p_{02} by an occurrence of t_{12}, and K_2 tokens are set in p_{21}, after which the occurrences of t_{02} begin. When p_{21} becomes unmarked, the occurrence of t_{21} moves the token from p_{02} to p_{01} and also resets p_{12} to K_1.

For traffic intensity equal to 0.9, the average waiting times (obtained by simulation of the model shown in Figure 4) are:

stage	*average waiting time*
1	24.805
2	4.009
3	3.381
total	32.195

This result is compared with a modified system, in which the leaky bucket is used at the front of the sequence of service stations, as shown in Figure 6.

This result is compared with a modified system, in which the leaky bucket is used at the front of the sequence of service stations, as shown in Figure 6.

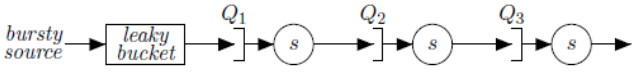

Figure 6. Queueing example with leaky bucket

For the leaking time equal to 0.30 time units, the waiting times are (stage "0" is the leaky bucket):

stage	waiting time
0	0.681
1	21.777
2	3.739
3	2.415
total	28.612

When the traffic intensity approaches 1, the differences are more significant (as the queueing becomes more intensive). For traffic intensity equal to 0.95, the average waiting times in the original system (Figure 4) are:

stage	waiting time
1	69.933
2	8.568
3	6.926
total	85.427

while the waiting times in the modified system (Figure 6) are:

stage	waiting time
0	1.954
1	33.229
2	5.052
3	4.921
total	45.156

4. Token Bucket Scheme

Token bucket algorithm is based on an analogy of a fixed capacity bucket, into which tokens are added at a fixed rate until the bucket is full. Each token usually represents a packet or a number of bytes in the packet (so several tokens may be needed to match a packet). When a packet is to be processed, the bucket is inspected if it contains the number of tokens matching the packet, and if this is the case, the tokens are removed from the bucket and the packet is sent forward. If the number of available tokens is insufficient, the packet can wait until more tokens are added to the bucket, or the packet can be dropped, or it is sent forward but is marked as being nonconformant, possibly to be dropped subsequently if the network is overloaded [13].

A conforming flow contains thus traffic with an average rate up to the rate at which tokens are added to the bucket.

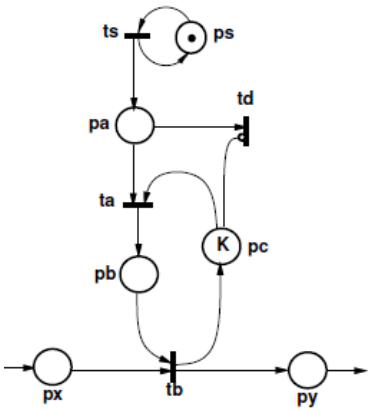

Figure 7. Petri net model of token bucket

Petri net model of the token bucket scheme, for the case when each packet is matched by a single token, and packets which do not have matching tokens are not dropped but are delayed, is shown in Figure 7.

Place p_b is the bucket of tokens. It p_b is marked, an arriving packet is forwarded by transition t_b moving one token from p_b to p_c; p_c is complementary to p_b and indicates the number of tokens missing in the bucket. Transition t_s with deterministic occurrence time T_s generates tokens with rate $1/T_s$. If the bucket is not full (i.e., p_c is marked), the generated token (in p_a) is moved to the bucket p_b by transition t_a. If the bucket is full, transition t_d discards the generated token.

The capacity of the bucket controls the burstiness of the shaped traffic. If this capacity is equal to K, at most K arriving packets are forwarded without additional delay while subsequent packages are affected by the token generation rate ($1/T_s$).

Token bucket shown in Figure 7 controls the number of packets but does not take the length of packets into account. This can be illustrated by the following example. Let the traffic be composed of two types of packets, packets of length 1 and packets of length 3. Let α denote the fraction of the total number of packets which are packets of length 3 ($0 \le \alpha \le 1$). If the length of packets is not taken into account, the traffic can be controlled only at the average level of packet length. However, if this average length changes, the traffic intensity can change quite significantly.

Figure 8 shows channel utilization as a function of α, the fraction of the total number of packets which are packets of length 3. For traffic control that does not take packet length into account, channel utilization changes (in Figure 8) from 0.25 (for $\alpha = 0$, i.e., traffic composed exclusively of packets of length 1) to 0.75 (for $\alpha = 1$, i.e., traffic composed exclusively of packets of length 3). It is rather straightforward to observe that with the same arrival rate of packets, the bandwidth required for $\alpha = 0$ is three times smaller than the bandwidth required for $\alpha = 1$.

It should also be observed that since the leaky bucket scheme does not take packet length into account, the effects of packet length on bandwidth requirements for token bucket without packet length control are very similar to those for leaky bucket.

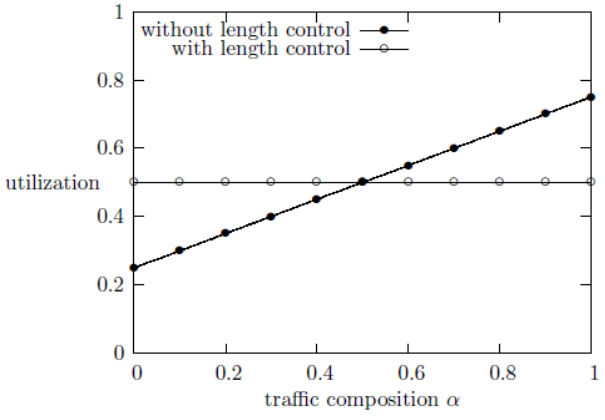

Figure 8. Channel utilization as a function of α

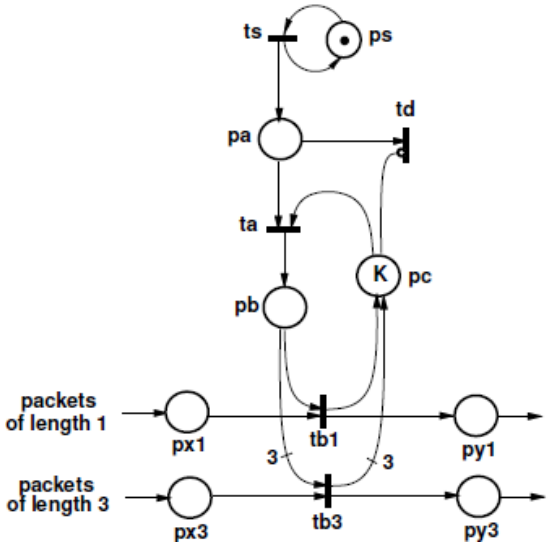

Figure 9. Petri net model for token bucket with packet length control

Packet length can be taken into account by assigning each token to a number of bytes in the forwarded packet and removing (from the bucket) the number of tokens representing the length of current packet (smaller number of tokens for shorter packets and larger numbers of tokens for longer packets). Figure 9 shows the token bucket scheme for two types of packets, one of length 1 and the other of length 3. These two types of packets enter the scheme through input places p_{x1} and p_{x3}, respectively. Each packet of length 1 is forwarded if there is at least 1 token in the bucket (otherwise it waits in p_{x1}); each packet of length 3 needs at least 3 tokens in the bucket to be forwarded (arcs (p_b, t_{b3}) and (t_{b3}, p_c) with weights 3). If there is insufficient number of tokens in the bucket, the packet is delayed (in p_{x1} or p_{x3}).

Figure 8 also shows channel utilization as a function of α for the case when packet length is taken into account. This utilization is practically constant at the level of 0.5. This level is determined by the rate of token generation in Figure 9 (i.e., by the occurrence time of transition t_s).

5. Traffic Policing

Traffic policing, i.e., enforcing conformance of the traffic to some requirements, restricts the flow of packets by either dropping the nonconformant packets or marking them as nonconformant so they can be dropped later. This can be illustrated by adding the traffic shaping schemes to the (strict) priority scheduling [14], with its well known possibility of blocking lower priority traffic by intensive traffic of higher priorities.

Figure 10 shows a simple priority queueing with two classes of traffic.

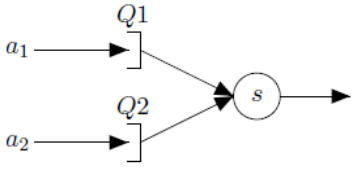

Figure 10. Priority queueing

Class 1 packets take priority in accessing the transmission channel (i.e., the server in Figure 10) over class 2 packets, which means that class 2 packets can be transmitted only when there are no class 1 packets waiting for transmission.

Petri net model for priority queueing is shown in Figure 11.

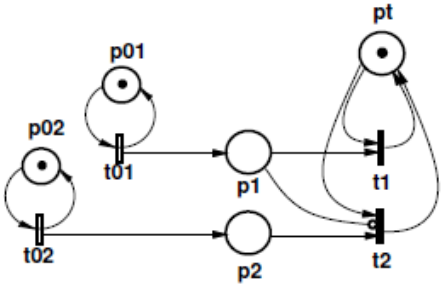

Figure 11. Petri net model of a priority queueing

In Figure 11, transition t_{01} with place p_{01} generate class 1 packets, and transition t_{02} with place p_{02} class 2 packets. If the occurrence times of t_{01} and t_{02} are exponentially distributed, the packets have exponentially distributed inter-arrival times; if the occurrence times are constant, the arrivals are deterministic, and so on. Transitions t_1 and t_2 represent the shared communication channel (the server in Figure 10) and place p_1 (with a single token) guarantees that only one packet can be transmitted at a time. The inhibitor arc (p_1, t_2) provides the priority of class 1 over class 2, i.e., packets from p_2 can be transmitted only when p_1 is unmarked (i.e., there are no class 1 packets waiting for transmission).

The behavior of (strict) priority scheduling is illustrated in Figure 12 by showing the utilization of the shared channel by class 1 and class 2 packets as functions of class 1 traffic intensity, ρ_1, with fixed traffic intensity of class 2, $\rho_2 = 0.4$.

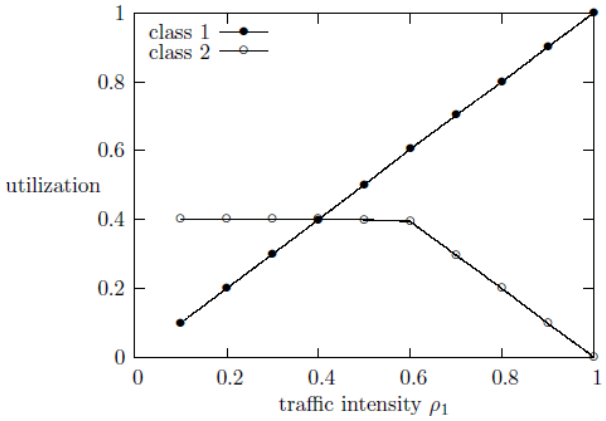

Figure 12. Channel utilization for (strict) priority queueing

As ρ_1 increases, the traffic in class 2 is not affected until $\rho_1 = 0.6$, when the channel becomes fully utilized ($\rho_1 + \rho_2 = 1$). Further increases in channel utilization by class 1 can be only achieved by blocking traffic in class 2; Figure 12 shows that class 2 traffic is gradually reduced to zero as ρ_1 increases.

It should be observed that in more realistic examples, with many classes of traffic, the blocking effects are more complex as they correspond to different interactions of traffic in classes of higher priorities.

The blocking of traffic can be restricted by using traffic shaping schemes. Figure 13 shows a modification of priority queueing in which leaky bucket is used to restrict class 1 traffic, and to prevent excessive blocking of class 2 traffic. The level of restriction is determined by the leaking time of the scheme used.

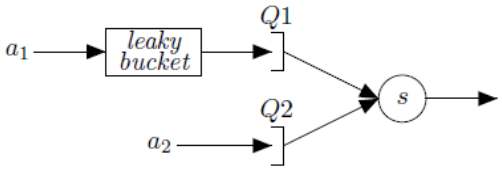

Figure 13. Priority queueing with leaky bucket

Petri net model of this modified priority queueing (Figure 13) is obtained by replacing place p_1 in Figure 11 by the model shown in Figure 2.

The effect of traffic policing by leaky bucket is shown in Figure 14 (which should be compared with Figure 12). Traffic restriction introduced by the leaky bucket is set at the level of 80%, i.e., traffic intensity for class 1 cannot exceed 0.8 with the remaining 20% of channel bandwidth "reserved" for class 2 traffic.

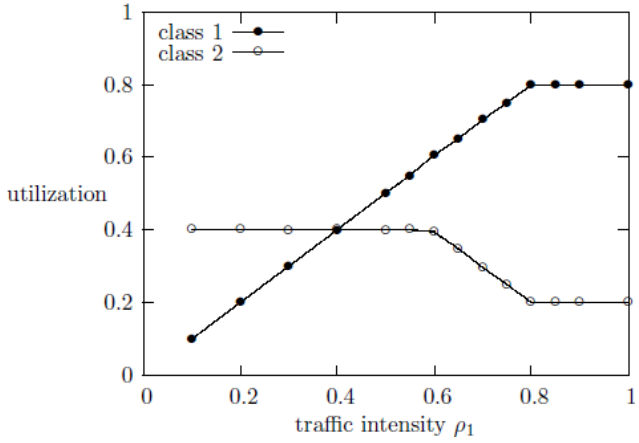

Figure 14. Channel utilization for priority queueing with leaky bucket

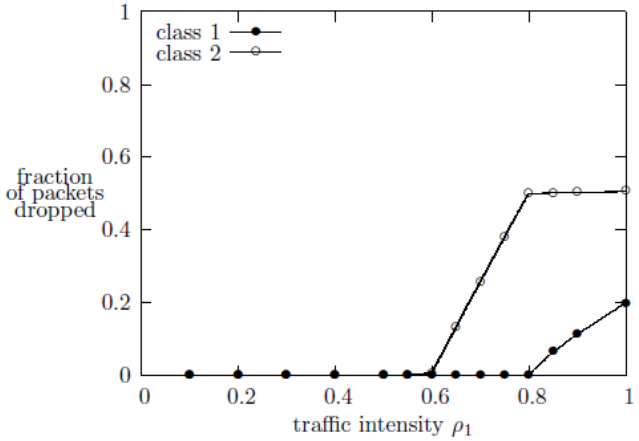

Figure 15. The fraction of packets dropped as a function of traffic intensity ρ_1

The restriction of the traffic introduced by leaky bucket results in dropping the packets which cannot be forwarded for transmission. Figure 15 shows the fraction of the total number of arriving packets that are dropped because they do not conform to the assumed traffic profile. As the traffic in class 2 is restricted from the level of 0.4 to 0.2 (by ρ_1 changing from 0.6 to 0.8), the fraction of packets dropped increases from 0 to 0.5 (i.e., 50%). Similarly, as channel utilization for class 1 cannot exceed 0.8, for $\rho_1 > 0.8$ all packets which cannot be transmitted are dropped.

It should be noted that practically the same traffic restriction obtained by using leaky bucket (Figure 14) can be obtained by using token bucket.

6. Concluding Remarks

Traffic shaping and policing can be done at the source, prior to entrance into the network, or within the networks [1]. If it is done at the source, it means a kind of self-regulation in order to ensure conformance to the traffic contract. Such conformance is desirable to minimize the amount of traffic disregarded at the network nodes.

The idea of traffic control within the network is to smooth the traffic and to reduce the queues and hence to incur shorter queueing delays and less delay jitter [15]. Since traffic shaping is done by buffering, some queueing delay is added during this process, but the smoothed traffic usually quickly compensates this additional delay. Overall, network performance is improved by smooth traffic [2].

Intelligent traffic control schemes can provide a particular quality of service (measured in jitter, packet loss, and latency) for an application or a user while still allowing other traffic to use all remaining bandwidth.

Traffic shaping is especially effective in busy networks, i.e., when the network traffic is close to the network capacity.

Finally, it should be noted, that the discussed methods are just basic elements of complex computer networks and that the behavior of real systems is very dynamic and usually difficult to predict. Therefore good understanding of the basic elements is needed to use the networks in an efficient and predictable way.

Acknowledgements

The Natural Sciences and Engineering Research Council of Canada partially supported this research through grant RGPIN-8222.

References

[1] Chen, T.M. (2007). "Network traffic management"; in: *Handbook of Computer Networks*, Bidgoli, H. (ed.), New York, NY: Wiley.

[2] Elwalid, A., Mitra, D. (1997). "Traffic shaping at a network node: theory, optimum design, admission control", Proc. IEEE INFOCOM'97, 444-454.

[3] Allen, A.A. (1991). *Probability, Statistics and Queueing Theory with Computer Science Applications* (2 ed), San Dego, CA: Academic Press.

[4] Jain, R. (1991). *The art of computer systems performance analysis*, New York, Y: J. Wiley & Sons.

[5] Flach, T., Papageorge, P., Tersiz, A., Pedrosa, L.D., Cheng, Y., Karim, T., Bassett, E.K., Govindan, R. (2016). "An internet-wide analysis of traffic policing", Proc. SIGCOMM'16, Florianopolis, Brazil.

[6] Murata, T. (1989). "Petri nets: properties, analysis and applications", Proceedings of IEEE, 77(4), 541-580.

[7] Reisig, W. (1985). *Petri Nets - an Introduction* (EATCS Monographs on Theoretical Computer Science 4), New York, NY: Springer-Verlag.

[8] Zuberek, W.M. (1991). "Timed Petri nets – definitions, properties and applications", Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models), 31(4), 627-644.

[9] Zuberek, W.M. (1987). "D–timed Petri nets and modelling of timeouts and protocols", Transactions of the Society for Computer Simulation, 4(4), 331-357.

[10] Zuberek, W.M. (1986). "M–timed Petri nets, priorities, preemptions, and performance evaluation of systems"; in: *Advances in Petri Nets 1985* (Lecture Notes in Computer Science 222), 478-498, Berlin, Heidelberg: Springer–Verlag.

[11] Swarna, M., Ravi, S., Anand, M. (2016). "Leaky bucket algorithm for congestion control", Int. Journal of Applied Engineering Research, 11(5), 3155-3159.

[12] Tannenbaum, A.S. (2003). *Computer Networks* (4 ed), Englewood Cliffs, NJ: Prentice-Hall.

[13] Tsai, T-C., Jiang, C-H., Wang, C-Y. (2006). "CAC and packet scheduling using token bucket for IEEE 802.16 networks", Journal of Communications, 1(2), 30-37.

[14] Georges, P., Divoux, T., Rondeau, E. (2005). "Strict priority versus weighted fair queueing in switched Ethernet networks for time-critical applications", Proc. 19-th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 141-145.

[15] Rexford, J., Bonomi, F., Greenberg, A., Wong, A. (1997). "Scalable architecture for integrated traffic shaping and link scheduling in high speed AT- M switches", IEEE Journal on Selected Areas in Communication, 15, 938-950.