A new approach to Service Differentiation: Non-Congestive Queueing

L. Mamatas and V. Tsaoussidis
Dept. Of Electrical & Computer Engineering
Demokritos University of Thrace, Greece
{emamatas, vtsaousi}@ee.duth.gr

Abstract

We evaluate a new approach to service differentiation, based on distinguishing congestive and non-congestive data. We initially define as non-congestive data only minimal-size packets, which do not contribute to congestion themselves, but, however, do suffer delays caused by larger packets. Case examples of such data are typical sensor data. Non-Congestive Queuing (NCQ) needs not follow a state-full approach to traffic differentiation. We show that (i) non - congestive data gets a much better service from the network, (ii) congestive data suffers no extra delays; instead, both fairness and efficiency are occasionally improved due to lower contention. Having more flows finishing early may lead to better resource sharing.

1 Introduction

Typical scheduling paradigms of packet networks do not match well the requirements of non-congestive applications, which transmit minor data volumes but suffer, however, major queuing delays. Sensor applications are some recent examples that demand a new type of service differentiation. Such applications do not really cause significant delays, raising naturally the issue of whether they deserve a prioritized service or not. For example, a sensor-generated packet may experience zero delay favored by a prioritized scheduling scheme, at a zero cost to other congestive flows. We attempt to exploit this idea in the context of computer networks, and in terms of correctness and practicality. We introduce a new paradigm for service differentiation, based on the principle of packet contribution to queuing delay. That is, we distinguish two traffic classes, congestive and non-congestive; non-congestive traffic gets prioritized service for two reasons: first, a non-congestive flow does not cause significant delay and hence needs not suffer from delays. Second, since it does not cause delays its service time should eventually have little impact on other flows and therefore, it may well get service first. The whole issue therefore is essentially turned down to exploiting the trade-off itself. How much is the benefit for the non-congestive applications and how much is the cost for the congestive ones?

Our primary assumption is that non-congestive traffic is generated in the form of small and periodic packet transmission. Certainly, typical sensor applications do fall into that category; however, other applications may fall into this category as well, if we judge solely on the basis of packet length. We discuss this issue later in this paper. However, the concept of service differentiation based on congestive and non-congestive data is powerful indeed: no flow states are required and no packet marking either.

Although the idea sounds straightforward, the system properties and design details reveal interesting dynamics. We first show with simple scenarios that the benefit for non-congestive flows is impressive. In terms of system service, this is translated into increased user satisfaction due to the increased amount of applications finishing much earlier. Notably, system performance is not degraded. Total time of applications system-wide is decreased. Furthermore, the gained performance of non-congestive flows is not always counterbalanced by a corresponding performance loss of congestive flows.

Although the proposed algorithm is still in experimental phase we do provide feedback on several issues: What amount of traffic should get prioritized service? Is there a performance loss when no congestive flows exist? Is there a loss when no non-congestive flows exist? However, more details need to be parameterized and evaluated and further analysis is on the works. At this stage of research, our results clearly indicate the applicability of non-congestive queuing (NCQ) for packet networks.

In section 2 we discuss the related work and detail our proposal. In section 3, we provide the pseudo-code of NCQ and present the basic assumptions, fundamental concepts and projected outcome. In section 4 we discuss the evaluation plan and metrics along with a justification for the plan. Furthermore, we present the results, analysis and justification. We show the impact of the percentage of packets that get prioritized service on system performance. In section 5 we discuss several open issues and address reasonable concerns. In section 6 we summarize our conclusions and future work.

2 Related Work

Service differentiation in computer networks is a topic of research, mainly focused on supporting application requirements for delay and bandwidth. Differentiation is mainly an operation that deals with reallocating bandwidth and controlling delay for the benefit of strict-service-requiring applications and at the cost of more flexible applications. Therefore, service differentiation is based on the principle 'get better service - if you need better service'.

Internet service differentiation has not been designed on the basis of theoretical research; rather, it was driven by the need for supporting real-time multimedia applications over the Internet. Such applications do have strict bandwidth and delay requirements; the flows that generate can describe their specifications in broad or detailed terms, and the network can plan for guaranteed service or otherwise for somewhat better service. Two approaches have gained acceptance recently: According to diffserv [1], the inherent properties of packet-switched Internet are masked with a number of gentle mechanisms, naturally matching Internet's structure, which - one way or another - shape traffic at some functionally-enhanced nodes. Alternatively, with intserv [2] the architecture itself can be redesigned to allow for guarantees through signaling and reservation.

Differentiation in both the aforementioned cases is application-specific and naturally oriented either by some explicit and strict flow characteristics or by some application class. Even in the latter case, associating application types with service classes requires a rather sophisticated implementation, ranging from packet marking, to shaping, scheduling and dropping schemes. Perhaps network engineering would have been different had the pressing demand of application requirements been ignored. For example, a natural principle to lead the design of network services (and consequently the service differentiation policy) could have been the network ability to function, the number of users serviced better without damaging the rest, or the service offered on the basis of the cost to other applications. It is not unnatural to service first applications that require minimal time for service; in that case the gain for such applications can be significant, while the cost for the other applications may be small.

A similar scheduling discipline has been studied in operating systems where some schedulers select processes based on their completion time, rather than the time they started (shortest job first). Such a service alone may lead to starvation in case the rate of small processes is sufficient to keep the processor busy; processes demanding more time for completion could never get their turn. However, due to the cost of context switch and the limited concurrent presence of processes, this domain had limited scheduling flexibility; our service differentiation scheme should guarantee not only better service for non-congestive data but also a non-degraded service to congestive applications. Thus, only a limited amount of non-congestive data should be able to benefit from the differentiated service; once the cost of better service to some flows impacts the service to other flows, service differentiation should terminate.

A lot has been done in the networking community aiming at controlling traffic based on its characteristics. Controlling is implemented either through scheduling or through dropping policies mainly aiming at penalizing high - bandwidth - demanding flows rather than favoring low - bandwidth - demanding flows. In [5] Floyd and Fall introduced mechanisms based on the identification of high-bandwidth flows from the drop-history of RED. The RED-PD algorithm (RED with Preferential Dropping) [7] uses per-flow preferential dropping mechanisms. Two other approaches that use per-flow preferential dropping with FIFO scheduling are Core-Stateless Fair queuing (CSFQ) [12] and Flow Random Early Detection (FRED) [6]. CSFQ marks packets with an estimate of their current sending rate. The router uses this information in conjuction with the flow's fair share estimation in order to decide whether a packets needs to be dropped. FRED does maintain a state although only for the flows which have packets in the queue. The flows with many buffered packets are having an increased dropping propability.

The CHOKe mechanism [10] matches every incoming packet against a random packet in the queue. If they belong to the same flow, both packets are dropped. Otherwise, the incoming packet is admitted with a certain propability. However, a high-bandwidth flow may have only a few packets in the queue. Authors of [9] continued the CSFQ and CHOKe approaches. Their proposed mechanism keeps a sample of arriving traffic. A flow with several packets in the sample, has an increased dropping probability. The Stochastic Fair Blue (SFB) [4] uses multiple levels of hashing in order to identify high-bandwidth flows. As the authors state, their mechanism works well only with a few high-bandwidth flows. Anjum and Tassiulas proposed in [3] a mechanism that drops packets based on the buffer occupancy of the flow while ERUF [11] uses source quench to have undeliverable packets dropped at the edge routers. On the other hand, SRED [8] caches the recent flows in order to determine the high-bandwidth flows.

3 Non-Congestive Queueing

NCQ is based on the principle 'Less Impact - Better Service' (LIBS). Although one can generalize the perspective to apply to all incoming packets, we only deal with two classes of packets: very small packets, deterministically set to 100 bytes and long packets that typical Internet applications use for data transfers. A natural question therefore is what if small-packet rate reaches levels, which delay significantly long-packet transmission. We complement the differentiating scheme with a service threshold: Non-congestive traffic cannot exceed a predetermined *ncqthresh* percentage of prioritized service. We investigate the impact of *ncqthresh* in the result section. NCQ uses priority queuing to implement priority service. That is, with in the same buffer, each packet is checked for its length, contrasted to a current priority rate and gets priority whenever it satisfies two conditions: (i) length is below 100 bytes and (ii) priority rate is below *ncqthresh*.

The algorithm below shows the pseudo-code for NCQ:

```
For every received packet
Begin
        Count received packets
        (congestive and non-congestive)
        if (packetLength<100)
         and
          (non-congestive packets /
           received packets < ncqthresh)
        then
         packet gets high priority
        else
          packet gets normal priority
        end
End
```

Clearly, applications that utilize small packet transfers at high rate may benefit from NCQ. As soon as the rate of priority reaches *ncqthresh*, no further prioritized service is allowed.

On the other hand, a typical application could be transformed into a small-packet application intentionally. Since the amount of gain is limited by the *ncqthresh* and the packet length, the transform should cause that much overhead and extended communication time that naturally the penalty of transformation will be greater than the gain.

4 Impact of NCQ

Initially we attempt to approach numerically the impact of NCQ priority on congestive traffic for any given proportion of traffic classes. We assume two classes of traffic (with Poisson arrival distribution). Class 1 is the non-congestive traffic and the class 2 the congestive. Class 1 has priority over class 2. We use a non-preemptive head-of-line priority system. The class 1 has smaller packets (and average service-time) and lower arrival rate.

We use the following notation:

Notation	Description
$\lambda_1 = ak$	arrival rate of Class 1
$\lambda_2 = k$	arrival rate of Class 2
$T_{S1} = br$	average service-time of Class 1
$T_{S2} = r$	average service-time of Class 2
$\lambda = \lambda_1 + \lambda_2$	total arrival rate
$u_1 = \lambda_1 T_{S1}$	utilization of Class 1
$u_2 = \lambda_1 T_{S1} + \lambda_2 T_{S2}$	cumulative utilization
T_{Q1}	queuing delay for Class 1
T_{Q2}	queuing delay for Class2
T_Q	average queuing delay

According to our model the total queuing delay per packet has two components: the waiting time and the service time.

Scenario 1: Priority Scheduling

We calculate the average waiting time for each of the two classes as:

$$T_{W1} = \frac{\lambda_1 T_{s1}^2 + \lambda_2 T_{s2}^2}{2(1-u_1)} \tag{1}$$

$$T_{W2} = \frac{\lambda_1 T_{s1}^2 + \lambda_2 T_{s2}^2}{2(1-u_1)(1-u_2)} \tag{2}$$

Consequently, the total average waiting time is the average of T_{W1}, T_{W1} weighted by the arrival rate for each class.

T_W = total average wait = $\frac{\lambda_1}{\lambda} T_{W1} + \frac{\lambda_2}{\lambda} T_{W2}$

We apply this to each class and we calculate afterwards the weighted average in order to get the total average time-in-system:

$$T_{Q1} = T_{W1} + T_{S1} \tag{3}$$

$$T_{Q2} = T_{W2} + T_{S2} \tag{4}$$

$$T_Q = \frac{\lambda_1}{\lambda} T_{Q1} + \frac{\lambda_2}{\lambda} T_{Q2} \tag{5}$$

From equations (3), (4), (5) we get:

$$T_Q = \frac{k(2-kr+a^3b^3k^2r^2+a^2bkr(-kr+3b(-1+kr))}{2(1+a)(-1+abkr)(-1+k(r+abr))}$$
$$+\frac{a(b^2kr+kr(1-kr)+2b(1-3kr+k^2r^2)))}{2(1+a)(-1+abkr)(-1+k(r+abr))}$$

Scenario 2: Non-Priority Scheduling

Without a priority queue, the two classes (non-congestive and congestive) would have the same average waiting time. In such a case, the network utilization of the system is:

$$u_{wp} = u_1 = u_2 = \lambda_1 T_{S1} + \lambda_2 T_{S2} \tag{6}$$

Service time:

$$T_{Swp} = \frac{\lambda_1}{\lambda} T_{S1} + \frac{\lambda_2}{\lambda} T_{S2} = \frac{k(1+ab)}{1+a} \tag{7}$$

For the average waiting time:

$$T_{W1wp} = T_{W2wp} = \frac{u_{wp} T_{Swp}}{2(1-u_{wp})} \tag{8}$$

For the average time-in-system:

$$T_{Q1wp} = T_{W1wp} + T_{S1} \tag{9}$$

$$T_{Q2wp} = T_{W2wp} + T_{S2} \tag{10}$$

From (9) and (10), we get:

$T_{Qwp} = \frac{\lambda_1}{\lambda} T_{Q1wp} + \frac{\lambda_2}{\lambda} T_{Q2wp}$
$= \frac{(1+ab)k(-2+k(r+abr))}{2(1+a)(-1+k(r+abr))}$

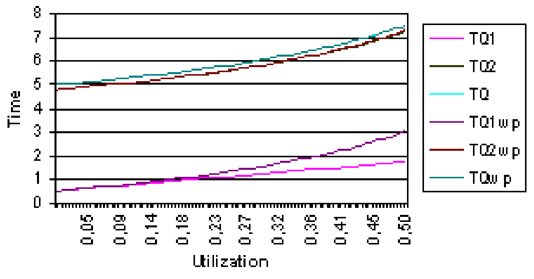

Figure 1. Numerical Results (5% non-congestive)

In Figure 1, we present the results of a simple scenario. The 5% of arriving packets form the non-congestive traffic (class 1) and the 95% the congestive (class 2). The service times are 0.5ms and 5ms for class 1 and class 2 respectively. While the average time of congestive traffic and the total average time are not affected by the use of priority queuing, the non-congestive traffic is significantly favored. When we increase the rate of non-congestive packets to 25% (see Figure 2), there is an impact to the congestive traffic in high utilizations (above 0.3).

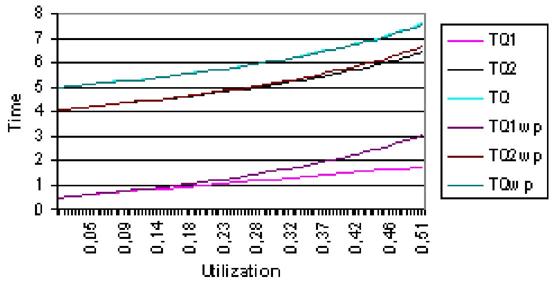

Figure 2. Numerical Results (25% non-congestive)

5 Evaluation

5.1 Evaluation Methodology

We have implemented our evaluation plan on the network simulator ns-2. We attempt to address three specific matters:

1. To show by simulation that numerical results do not lack any important parameter. For that, we used a simple dumbbell topology as shown in figure 3

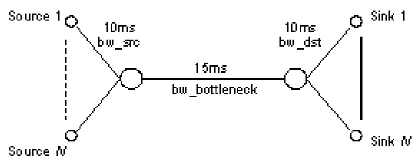

Figure 3. Simulation topology

and we measure:

$$Goodput = \frac{Original\_Data}{Time}$$

where $Original\_Data$ is the number of bytes delivered to the high-level protocol at the receiver (i.e. excluding retransmitted packets and overhead) and $Time$ is the amount of time required for the data delivery. We used the system Goodput in order to measure the overall system efficiency. The system Goodput is defined as:

$$SystemGoodput = \frac{\sum(Goodput_i)}{n}$$

where $Goodput_i$ is the Goodput of the i^{th} flow and n the flow number.

Furthermore, we experiment with variants of traffic thresholds and traffic class proportions, to demonstrate the overall system behavior when the ncq parameters change.

2. To exploit the impact of non-congestive queuing on time-sensitive, sensor-like data tranfers. For that, we used again a simple dumbbell, running however CBR traffic instead of ftp. In the same context, we measure performance using a real-time performance index, which we call Application Success Index defined as:

   $$ApplicationSuccessIndex = \frac{PacketsDeliveredinTime}{PacketsDelivered}$$

   where `PacketsDeliveredinTime` is the number of packets which have been received by the application in time and `PacketsDelivered` is the total number of packets received by the application.

   Additionally, we measure application efficiency and user satisfaction based on the `worst` and `average task completion time` as well as on the `number of completed tasks`. Every congestive FTP flow transmits 1Mb data and every non-congestive FTP flow 100Kb. We regard as a task the successful transmission of the carried data of a flow.

3. To exploit the potential applicability of NCQ on Gateways. We used a complex topology (see figure 4), where peripheral sources generate traffic to a specific gateway, crossing the traffic generated by other sources that follow a partially different path. The number of flows now increases (up to 1000 flows) to capture the circumstances of wider area network.

We analyze each scenario and corresponding results, in turn.

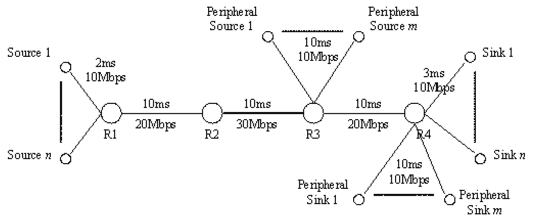

Figure 4. Network topology with multiple bottlenecks and cross traffic

5.2 Evaluation Results

In the following results we used *ncqthresh* 0.05. The non-congestive flows were the 5% of all flows (unless it is stated otherwise).

5.2.1 Scenario 1: Non-Congestive FTP flows and Dumbbell Topology

As we can see in Figure 5, the non-congestive flows have significant performance gains (4.9 times - in case of 70 flows) in terms of Goodput. Although non-congestive traffic is clearly favored by NCQ, we also notice occassionaly better performance for the congestive flows too (Figure. 6). This is not unreasonable: the impact of timeouts caused by short packets is more significant for non-congestive flows compared with the impact of long packets; i.e., regardless of the packet length, timeout is the same and extending total time for a small retransmission degrades system throughput. It is very interesting to note that the NCQ has a positive impact also in the system's average / worst time (Figures 8, 9). According to Figure 9, all tasks are completed up to 11 seconds sooner (in case of 30 flows).

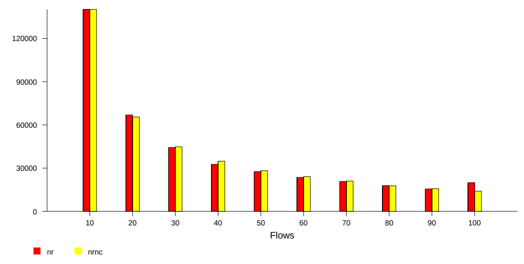

Figure 5. System Goodput

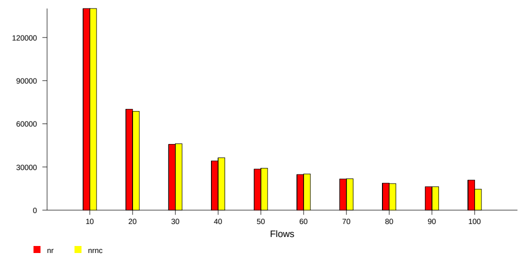

Figure 6. Average Goodput of Congestive Flows

We have also varied the traffic proportion of congestive and non-congestive flows. As the rate of the non-congestive flows increases (i.e. the number of the non-congestive packets increases), the benefit for favored non-congestive packets scales down. This behavior is not symptomatic. Instead, the `ncqthresh` value confines priority service for guarantee small impact on congestive flows. For example, we can see in Figure. 10 (rate of non-congestive packets 20%) that the NCQ algorithm favors only a small portion

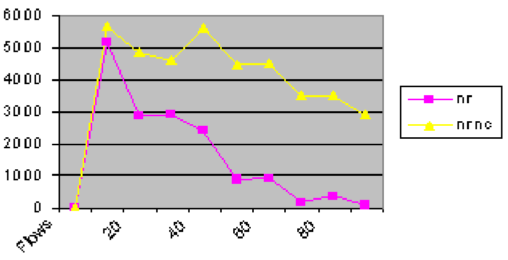

Figure 7. Average Goodput of Non-Congestive Flows

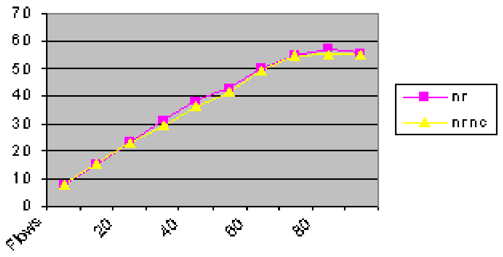

Figure 8. Average Time

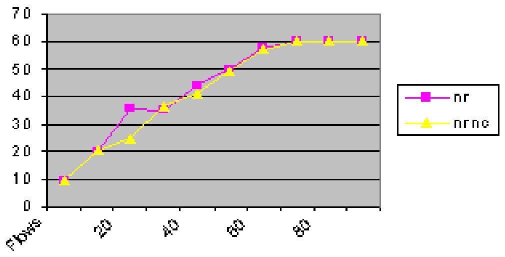

Figure 9. Worst Time

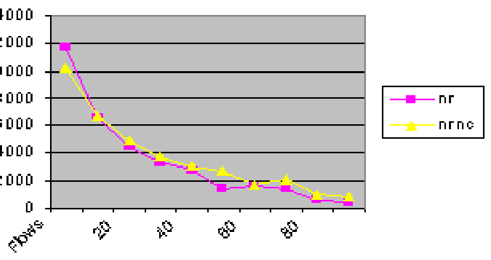

Figure 10. Average Goodput of Non-Congestive Flows (20% NC packets)

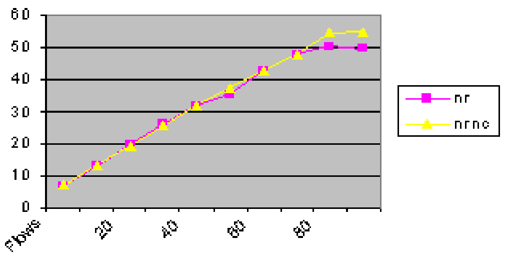

Figure 11. Average Time of Non-Congestive Flows (20% NC packets)

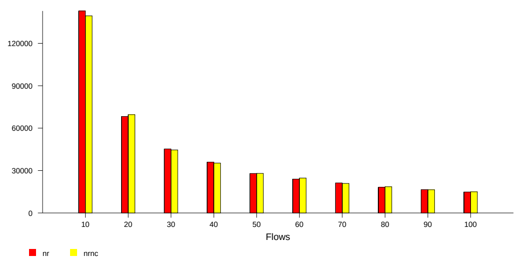

Figure 12. Average Goodput of Non-Congestive Flows (25% NC packets)

of non-congestive packets. This reduces performance gains and further, as the rate of non-congestive packets increases (25% - Figure. 11), NCQ impact is almost cancelled.

As we can see from Figure 13, NCQ leads to increased user satisfaction due to the increased number of applications finishing much earlier. Also, since flows are faced earlier with less contention, they seem to be able to share the channel better. However, this claim remains to be proven in future work.

Impact of traffic threshold (ncqthresh)

We carried out the same experiment with different values of *ncqthresh* (0.01, 0.03, 0.05). As we can see from Figure 14, in average, the value of 0.05 is a good choice. Additionally, Figure 14 indicates that increasing further the threshold does no guarantee better service. For example, in Figure 14 flows with *ncqthresh* 0.03 finish earlier.

5.2.2 Scenario 2: Non-Congestive CBR flows

In this scenario, we used CBR non-congestive flows (interval 0.1 sec, packet size 100 bytes). The non-congestive flows seem to be unaware of the increasing level of con-

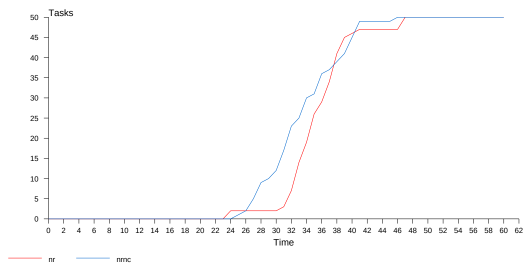

Figure 13. Tasks Completed (for 50 flows)

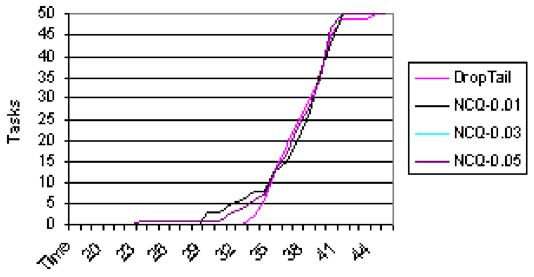

Figure 14. Tasks Completed and Traffic Treshold

tention (due to the congestive flows). The behavior of the system in terms of `application success index` (Figures. 16, 17, 18) demonstrates the applicability of NCQ for real-time applications (eg. time-sensitive periodic sensor data). The performance gains in terms of goodput were significant (Figure. 15).

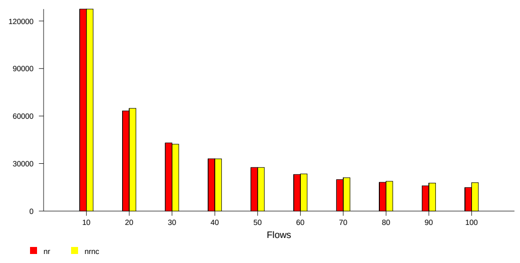

Figure 15. System Goodput

5.2.3 Scenario 3: Non-Congestive FTP flows and Complex Topology

We used a complex network topology with multiple bottlenecks and cross traffic (Figure. 4). Congestive flows form the main traffic, while non-congestive flows form the cross traffic.

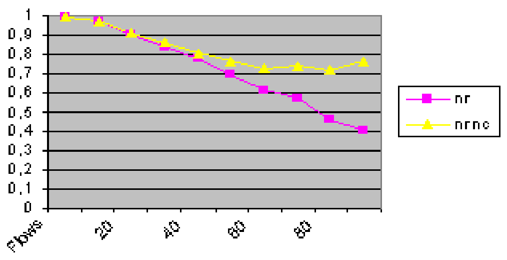

Figure 16. System's Application Success Index

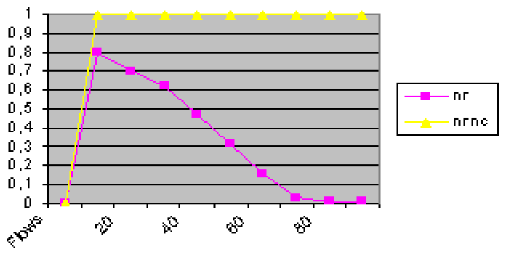

Figure 17. Non-Congestive Flows' Application Success Index

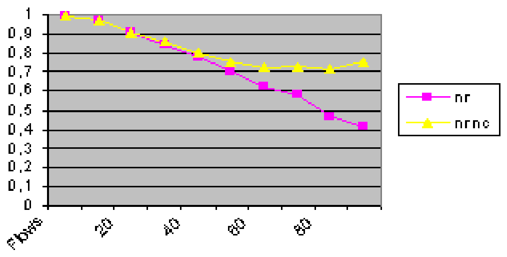

Figure 18. Congestive Flows' Application Success Index

Figures 19, 20, 21 depict the behavior of the system when the NCQ algorithm is used only in the gateway router (router R3). The number of participating flows now ranges from 100-1000 to reflect the different topology of a wide-area network.

6 Open Issues

Although we demonstrated NCQ's high potential, we do not presently address all concerns necessary to justify its

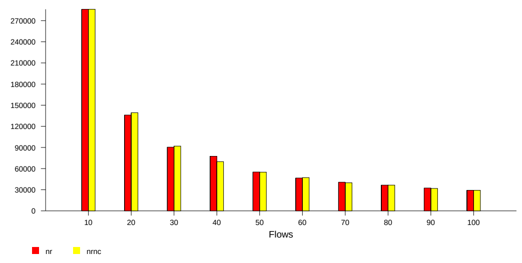

Figure 19. Average Goodput of All Flows

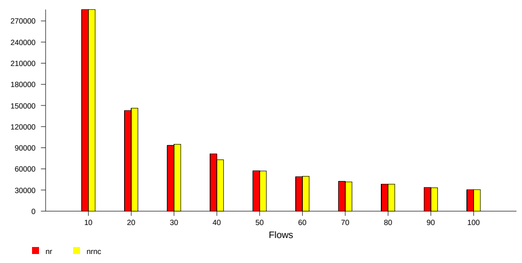

Figure 20. Average Goodput of Congestive Flows

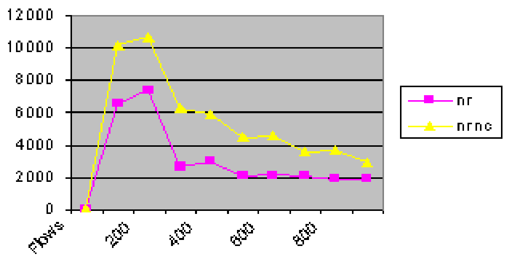

Figure 21. Average Goodput of Non-Congestive Flows

need for deployment. For example, a single application which utilizes small-size packets for data transmission may be congestive indeed. That means, such application will probably monopolize the priority service for themselves. We are working on an extension of the algorithm to assign probabilistically priority service to small packets. The probability per packet decreases as the rate of non-congestive packet exceeds the *ncqthresh*. Probabilistic priority will guarantee fairness among non-congestive flows in a similar fashion to RED's probabilistic dropping. Alternatively, the *ncqthresh* is dynamically adjusted, based on the projected outcome. In another front of research, acks and control packets may benefit from the priority treatment. Control packets prioritization are expected to boost the performance of short-lived flows (mice) increasing fairness compared to long-lived flows (elephants). Acks are expected to increase transmission rate; how far this can happen (considering also the delayed-ack scheme which is widely deployed) and how far it can impact congestion control is an open issue.

7 Conclusions

We have shown that NCQ is a simple but powerful tool for differentiation, particularly beneficial for applications that utilize small rates and short packets, such as typical sensor applications. We discovered that a limited prioritization has a dual impact: it benefits non-congestive flows significantly and reduces contention among the congestive flows, resulting occasionally in better delays and throughput for all applications. NCQ is still in experimental phase. Further analytical and experimental results with more sophisticated mechanisms are underway.

References

[1] Differentiated services charter. Technical report, Web Page: http://www.ietf.org/html.charters/diffserv-charter.html.

[2] Integrated services charter. Technical report, Web Page: http://www.ietf.org/html.charters/intserv-charter.html.

[3] F. M. Anjum and L. Tassiulas. Fair bandwidth sharing among adaptive and non-adaptive flows in the internet. In *IEEE InfoCom 99*, March 1999.

[4] W. Feng, D. Kandlur, D. Saha, and K. G. Shin. BLUE: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, April 1999.

[5] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[6] D. Lin and R. Morris. Dynamics of random early detection. In *SIGCOMM '97*, pages 127–137, Cannes, France, september 1997.

[7] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router, 2001.

[8] T. J. Ott, T. V. Lakshman, and L. H. Wong. SRED: Stabilized RED. In *Proceedings of INFOCOM*, volume 3, pages 1346–1355, 1999.

[9] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping, 2001.

[10] R. Pan, B. Prabhakar, and K. Psounis. CHOKe - A stateless queue management scheme for approximating fair bandwidth allocation. March 2000.

[11] A. Rangarajan. Early regulation of unresponsive flows. Technical Report TRCS99-26, July 1999.

[12] I. Stoica, S. Shenker, and H. Zhang. Core -stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *SIGCOMM*, pages 118–130, 1998.