# NCP: Finishing Flows Even More Quickly

Debessay Fesehaye, Pengye Xia[†],
P. Brighten Godfrey, Klara Nahrstedt[†] and Steven S. Lumetta[‡]
[†]Department of Computer Science, [‡]Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
{dkassa2,pxia3,pbg,klara,lumetta}@illinois.edu

## ABSTRACT

The transmission control protocol (TCP) is the major transport layer protocol in the Internet today. TCP and its variants have the drawback of not knowing the explicit rate share of flows at bottleneck links. The Rate Control Protocol (RCP) is a major clean slate congestion control protocol which has been recently proposed to address these drawbacks. RCP tries to get explicit knowledge of flow shares at bottleneck links. However, RCP under or over estimates the number of active flows which it needs to obtain the flow fair rate share. This causes under or over utilization of bottleneck link capacity. This in turn can result in very high queue length and packet drops which translate into a high average file completion time (AFCT).

In this paper we present the design and analysis of a Network congestion Control Protocol (NCP). NCP can give flows their fair share rates and hence resulting in the minimum AFCT. Unlike RCP, NCP can also use accurate formula to calculate the number of flows sharing a network link. This enables NCP to assign fair share rates to flows without over or under-utilization of bottleneck link capacities. Simulation results confirm the design goals of NCP in achieving minimum AFCT when compared with RCP.

## Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]; C.2.2 [**Network Protocols**]: Transport Protocol Design

## General Terms

Computer Systems Organization

## Keywords

Congestion control, protocol design, file download time

## 1. INTRODUCTION

The majority of network traffic uses the transmission control protocol (TCP) [6] as a congestion control protocol. TCP was very successful preventing congestion in the early stages of the Internet and before the emergence and vast expansion of other types of network and networking technologies. In spite of its success in reducing (avoiding) congestion in the early times of the Internet, TCP is now finding it increasingly difficult to cope with the growing Internet and network technologies [5, 9].

There have been numerous research efforts to deal with the weaknesses of the deployed congestion control (TCP). The current modifications to TCP such as HighSpeed TCP [4] inherit the main problems of TCP in not quickly knowing the bottleneck link share of flows. This results in flows taking longer to finish than necessary [2].

The eXplicit congestion Control Protocol (XCP) [7] is designed to achieve full link utilization and hence high per flow throughput. However XCP is not fair to short flows which are the majority of Internet flows resulting in higher average file completion time (AFCT) [2]. The Rate Control Protocol (RCP) [2] on the other hand was designed to finish flows quickly. But RCP under or over estimates the number of active flows which it needs to obtain the rate at which flows send packets. This can result in under or over utilization of bottleneck link capacity which in turn results in high queue length, packet drops and high AFCT.

In this paper we discuss the design and analysis of a Network congestion Control Protocol (NCP). The NCP approach derives a simple and effective congestion control rate metric which routers calculate and at which sources send data. Unlike TCP this rate metric can obtain the a very high link utilization and the low queue size and hence resulting in the very small AFCT. It can also be fair to all flows. NCP also uses an accurate derivation of the number of active flows and hence doesn't suffer from such estimation errors of RCP. NCP can be easily implemented using the OpenFlow [12] architecture.

Previous results [2] have shown how RCP outperforms XCP [7] and TCP. The simulation results we present in this paper show how NCP outperforms RCP.

The main contributions of this paper are as follows.

- We show that the performance of the Rate Control Protocol (RCP) degrades with network congestion.
- We propose a Network congestion Control Protocol

(NCP) which is a novel congestion control scheme that overcomes the weaknesses of RCP.

- We implemented NCP in the NS2 simulator and present results which show how NCP outperforms RCP. We have presented initial stability analysis of NCP.
- We have introduced a new resource sharing scheme called Efficient Sharing (ES). ES can be more efficient than the traditional Processor Sharing (PS) in utilizing unused network resources without the need of multiple queues and complex schemes for flows. We have shown that NCP is an ES protocol.

The rest of the paper is organized as follows. We first present the NCP algorithm in section 2. In section 3 we present the derivation of the NCP rate. Sections 4 and 5 show how NCP can achieve processor sharing and even more efficient sharing (ES) than the traditional processor sharing (PS) [8]. The NCP packet header format and NCP stability analysis are presented in sections 6 and 7. After validating the performance of NCP using simulation in section 8, we give a brief summary in section 9.

## 2. NCP ALGORITHM

The NCP algorithm at the end-hosts and at the routers can be described as follows:

- A source sends each byte $j$ with its desired rate $\hat{R}_j$.
- Each router in the network calculates $R(t)$ using equation 2 or equation 8 every control interval $d$, $0 < d \leq RTT_{max}$. Here $RTT_{max}$ is the maximum RTT of the flows which can be known or estimated offline.
- Each router in the path of a packet associated with byte $j$ checks if $R(t) < \hat{R}_j$ in which case it overwrites $\hat{R}_j$ and forwards it unchanged otherwise.
- The destination then copies the $\hat{R}_j$ in the data packet to the ACK packet.
- The source sets its current window size $w'_j = \hat{R}_j RTT_j$ upon receipt of the ACK packet.
- Each router updates its $R(t)$ value every control interval $d$.

## 3. NCP RATE

To define and derive the NCP rate metric, we first present descriptions of NCP parameters in table 1.

**Table 1: NCP Parameters**

| Parameters | Description |
|---|---|
| $C$ | Link capacity in bytes per sec |
| $d$ | Length of duration of control interval in sec |
| $q(t-d)$ | Queue size from the previous interval in bytes |
| $q(t)$ | Queue size from the current interval in bytes |
| $R(t-d)$ | Rate allocation of the previous interval in bytes |
| $R(t)$ | Rate allocation of the current interval in bytes |
| $N$ | Number of flows in the current interval |
| $L$ | Total number of bytes which arrive to the router during a control interval, $d$ |
| $\alpha, \beta$ | Stability parameters |

Given the notations in table 1, the per flow fair NCP rate allocation at a bottleneck router is derived as follows.

The intuition behind NCP is the assertion that the total number of bytes sent to a router (link) shouldn't exceed the bandwidth-delay product minus the queue size at the router. The total number of bytes sent to a router in the next interval is the sum of the *per byte cwnd* which is the number of bytes sent for each of the $w_j$ successfully transmitted bytes being sent in the current interval. Hence if $R_j = w_j/RTT_j$ denotes the rate associated with the $j$th of the $L$ bytes which arrive to the router,

$$\sum_{j=1}^{L} \frac{R(t)}{R_j} = \alpha C d - \beta q(t). \tag{1}$$

This implies that

$$R(t) = \frac{\alpha C d - \beta q(t)}{\sum_{j=1}^{L}(1/R_j)} = \frac{\alpha C - \beta \frac{q(t)}{d}}{\frac{1}{d}\sum_{j=1}^{L}\frac{1}{R_j}}. \tag{2}$$

After the initial allocation of the rate given by equation 2 at time $t-d$, unless new flows arrive, the sending rate, $R(t-d)$, is given by the same equation 2. Hence in such cases if we estimate $R_j$ with $R(t-d)$, equation 2 becomes the same as equation 8. The stability parameters $\alpha$ and $\beta$ are discussed in section 7.

Another derivation of the NCP fair share rate is also given in our previous work [3].

## 4. NCP CAN ACHIEVE PS

The inter-byte time $\sigma_j$ is defined as the time between two consecutive bytes for a flow associated with byte $j$. It is given by

$$\sigma_j = \frac{1}{R_j}. \tag{3}$$

Now suppose a router has seen $L$ bytes within the control time interval $d$. If $n_i$ of these bytes carrying $\sigma_i$ (in their corresponding packet) from source $i$ are received by the router during the control interval $d$, then taking the denominator of equation 2 we have

$$\frac{1}{d}\sum_{j}^{L}\frac{1}{R_j} = \frac{1}{d}\sum_{i=1}^{N} n_i \frac{1}{R_i} = \sum_{i=1}^{N}\frac{n_i}{d}\frac{RTT_i}{w_i} \tag{4}$$

where $N$ is the number of active flows and $w_i$ is the congestion window size (cwnd) of flow $i$ which is the number of bytes source $i$ sends during its round trip time ($RTT_i$). The variable $n_i$ is the total number of flow $i$ bytes which arrive at the router during the control interval $d$.

In the case where all bytes sent from a source $i$ at the rate of $R_i = w_i/RTT_i$ arrive to the next hop router (switch) at the same rate (as all the bytes of a flow to a router can be spaced at an equal interval of $\sigma_i$ on average) we have that

$$\frac{n_i}{d} = \frac{w_i}{RTT_i}. \tag{5}$$

This implies that $\frac{1}{d}\sum_{j}^{L}\frac{1}{R_j} = N$ which means that NCP can achieve PS.

In the next section we will discuss scenarios where $\frac{n_i}{d} < \frac{w_i}{RTT_i}$ and where NCP can perform better than the traditional processor sharing (ES) in a scheme we define as *efficient sharing (ES).*

## 5. NCP CAN ACHIEVE ES

In addition to achieving processor sharing (PS), NCP can also handle scenarios which a traditional PS scheme cannot handle. This enables NCP to achieve more efficient sharing (ES) than PS. We will next use two scenarios to describe this new concept which we define as *Efficient Sharing (ES)* .

### 5.1 Single Bottleneck Scenario

There can be a scenario where $\frac{n_i}{d} < \frac{w_i}{RTT_i}$. This happens for instance when a new bottleneck link is formed in the flow path before the location of the previous bottleneck link which allocated $R_i = \frac{w_i}{RTT_i}$ to flow $i$. The new bottleneck link then drops or delays packets of flow $i$ resulting in smaller rate $\frac{n_i}{d}$ arriving to the previous bottleneck link. In this case, NCP in the previous bottleneck link counts flow $i$ as less than one flow (*fractional flow*) which is equal to $\frac{n_i}{d}/R_i$. On the other hand the traditional PS counts each of such flows as one flow. In this case, the PS approach at the previous bottleneck link divides the capacity by more than the *actual* number of flows. This results in PS allocating less rates to some flows which need more. Dividing the capacity by the *exact fractional number* of flows, NCP however gives the capacity unused by some flows to flows which can use it without causing buffer overflow or resource underutilization. To do this, NCP doesn't require any special queues or complicated operations as the allocation is done using NCP rate equation. On the other hand the scenario where $\frac{n_i}{d} > \frac{w_i}{RTT_i}$ may occur for instance when bursts of packets of a flow arrive to a link. In this case NCP counts such flows as $\frac{n_i}{d}/R_i$ which is more than one flow. Hence NCP assigns less rate to flows to absorb the bursts of packets.

Another important result from equations 5 and 4 is that unlike RCP [2] and XCP [7] the estimation of the control interval, $d$ in NCP *doesn't need the flow RTTs.* It can be set to any reasonable value between maximum and minimum RTT values of flows obtained from experiments. The smaller the value of $d$ the more recent bottleneck rate values the packets carry back to their sources. Flow RTTs can also be used to obtain $d$ like XCP and RCP.

### 5.2 Multi-Bottleneck Network

In a network where different flows are bottlenecked at different links, some flows may not be able to utilize their equal share allocation at a link which is a bottleneck to other flows. If the bottleneck link allocation of flow $i$ is $R_i$ and if its current equal rate share at its none-bottleneck link is $R(t-d) > R_i$, then flow $i$ can waste its none-bottleneck link capacity which can otherwise be used by other flows bottlenecked at that link. This can result in NCP not achieving ES.

To deal with this scenario, NCP replaces the $R_j$ in the denominator of equation 2 with

$$max(R_j, R(t-d)) \tag{6}$$

where $R_j$ is the bottleneck rate carried by packet associated with byte $j$ of a flow and $R(t-d)$ is the rate allocation of flows at a link for the current interval.

NCP uses expression 6 only if the flow associated with byte $j$ is atleast in its second RTT sending packets at its bottleneck link rate. NCP can check this by comparing $R_j$ against the initial NCP rate $R_{init}$ of flows which can be known before hand as the ratio of initial *cwnd* and some average flow RTT. In this case if $R_j \leq R_{init}$ NCP doesn't use the expression 6 as the flow may be just starting. NCP packet header can also carry a single bit to indicate the start of a flow. If possible SYN packet can also be used to indicate the start of the flow.

Here is some explanation of why the approach in expression 6 can achieve ES (Efficient Sharing). If $R_j < R(t-d)$, then a flow which owns byte $j$ should be treated as a partial (fractional) flow by the router which allocated $R(t-d)$ to the flows (including the flow of byte $j$). This enables NCP to assign the unused resource to other flows bottlenecked at that router.

On the other hand if $R_j > R(t-d)$ then NCP achieves ES by treating the flow of byte $j$ as at least one flow as it can cause temporary queue spikes (being late to learn its new allocation). This occurs for instance because the allocation $R_j$ was much older than $R(t-d)$ as the flow has an RTT too long (longer than the control interval) to know about its latest rate allocation.

The idea of ES is different from generalized processor sharing (GPS) [13] approach. In GOS flows get a certain weighted share of the resource and hence benefit at the expense of others. The ES doesn't penalize some flows to benefit other flows. In fact we can also have a Generalized ES (GES) where the resource is used according to some weights as is the case of GPS.

If we approximate $R_j$ used in equation 2 with $R(t-d)$ even if $R_j > R(t-d)$, we get

$$\begin{aligned} N_a &= \frac{1}{d}\sum_j^L \frac{1}{R_j} \approx \frac{1}{d}\sum_j^L \frac{1}{R(t-d)} \\ &\approx \frac{1}{R(t-d)}\frac{L}{d} = \frac{y(t)}{R(t-d)} \end{aligned} \tag{7}$$

where $y(t) = \frac{L}{d}$ is the total input traffic rate in bytes during the control interval $d$ at the router.

When NCP uses equation 7, it can overestimate the actual number $N_a$ of flows when $\frac{1}{R_j} < \frac{1}{R(t-d)}$. This overestimation of $N_a$ can result in a lower rate allocation to all flows which in turn can result in link underutilization.

In a simplified and efficient version of NCP, we also use equation 7 in the denominator of equation 2 as an estimation of the actual number of flows. The resulting simplified NCP rate is then given by

$$R(t) = \frac{\alpha C d - \beta q(t)}{dN_a} \tag{8}$$

The derivation in equation 7 shows that the main strength of this simplified version of NCP lies on its use of the *fractional flow* concept where flows can be counted as partial flows unlike the case of PS. Hence the simple expression given by equation 7 is a very good estimator of ES. This implementation allows NCP packet header to be even smaller (about 8 bytes) as shown in section 6. In the simulation experiments of this paper we used the exact NCP rate given by equation 2.

## 6. NCP PACKET HEADER FORMAT

The NCP protocol can have two packet header implementation schemes. The first one which is shown in figure 1 has a 12 byte header.

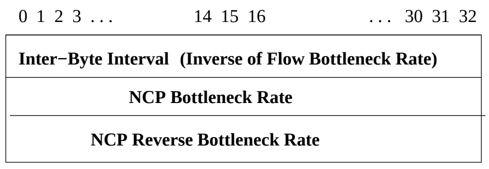
0 1 2 3 . . .
14 15 16
. . . 30 31 32
Inter−Byte Interval (Inverse of Flow Bottleneck Rate)
NCP Bottleneck Rate
NCP Reverse Bottleneck Rate

**Figure 1: NCP header with 12 bytes**

The first field is the *Inter-Byte Interval* length $\sigma_j = 1/R_j$, where $R_j$ is the current sending rate attached to a packet associated with byte $j$ of the corresponding flow. The routers in the path of byte $j$ (its associated packet) use this field to obtain the NCP rate given by equation 2. The second field is the *NCP Bottleneck Rate* $\hat{R}_j$ which is the rate initialized to be the desired rate by source. The bottleneck router in the path of the packet associated with byte $j$ can then overwrite the value. This rate is the minimum of all the rates in the path of the packet associated with byte $j$. The third field is *NCP Reverse Bottleneck Rate* which is the same NCP bottleneck rate which the receiver copies to its outgoing packets (ACK packets for example). The simulation results for NCP used in this paper use this implementation scheme of the NCP header.

The second implementation scheme of the NCP header is shown in figure 2 is without the $\sigma_j$ field. This implementation can reduce the NCP packet header to 8 bytes.

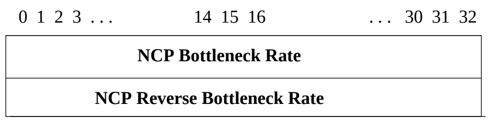

**Figure 2: NCP header with 8 bytes**

In this implementation scheme each source sets the value of the *NCP bottleneck rate* ($\hat{R}_j$) to its desired rate. Each router in the path of the packet associated with byte $j$ calculates the rate using equation 8. If this rate is smaller than the $\hat{R}_j$ in the packet header, then the router replaces the $\hat{R}_j$ in the packet header with what it obtain using equation 8. The receiver then copies the value of the *NCP bottleneck rate* value which routers may have changed into the ACK (returning) packets. The receiver of the ACK packets then adjusts its *cwnd* to the product of the rate it gets from the ACK packets and its RTT.

## 7. STABILITY ANALYSIS

In this section we present stability analysis using control theory. We have also previously done some initial stability analysis using the fixed point theory as shown in [3].

### 7.1 Lyapunov Stability

The rate allocation by NCP queue at a bottleneck router is done every control interval $d$. This allocation is received by each source sharing the bottleneck link after a round trip time (of each of the sources). This new rate allocation changes the congestion window $w_j$ of each source $j$. So the aggregate feedback sent per unit time is the sum of the derivatives of the congestion windows. This feedback is similar with the XCP feedback and hence we have

$$\sum_j \frac{dw_j}{dt} = C - \Lambda(t-d) - \frac{q(t-d)}{d} \qquad (9)$$

where $\Lambda(t-d)$ and $q(t-d)$ are the total arrival rate and queue size in the previous control interval and $C$ is the link capacity.

Adding the control parameters $\alpha$ and $\beta$ for stability, Equation 9 becomes

$$\sum_j \frac{dw_j}{dt} = \alpha(C - \Lambda(t-d)) - \beta\frac{q(t-d)}{d}. \qquad (10)$$

As shown in [1] and [10], the NCP feedback mechanism is given by the delay differential equations

$$\begin{aligned} \Lambda'(t) &= \frac{\alpha}{d}(C - \Lambda(t-d)) - \frac{\beta}{d^2}q(t-d) \\ q'(t) &= \begin{cases} \Lambda(t) - C, & q(t) > 0 \\ max\{\Lambda(t) - C, 0\}, & q(t) = 0. \end{cases} \end{aligned} \qquad (11)$$

As the NCP feedback mechanism can be written in Equation 11, appropriate Lyapunov functions can be used to find stable values of the control parameters $\alpha$ and $\beta$. For instance the work [10] shows that $\beta/d^2 = \alpha/d$ gives stability. This for instance implies that if $\alpha = 1.0$ , $\beta = d$. Previous work [1] also shows a wide range of stable values for protocols whose feedback mechanism can be written in the form of Equation 11. Our detailed simulation results also show that $\alpha = 1 = \beta$ gives stable values for NCP. We are also working on using the Lyapunov functions in [10] to find even wider stable regions for $\alpha$ and $\beta$.

## 8. SIMULATION ANALYSIS

In previous studies [2], RCP was shown to outperform TCP and XCP. In this section we evaluate the performance of NCP comparing it with RCP using NS2 [11] which is the state of the art network simulator.

To validate the performance of NCP, we implemented the NCP source as a sub-class of TCP-Reno and NCP queue as a subclass of DropTail Queue in NS2. In our experiments, packets are spaced by an inter-packet time which is equal to the inverse of the rate allocation of the flow received from the ACK packets.

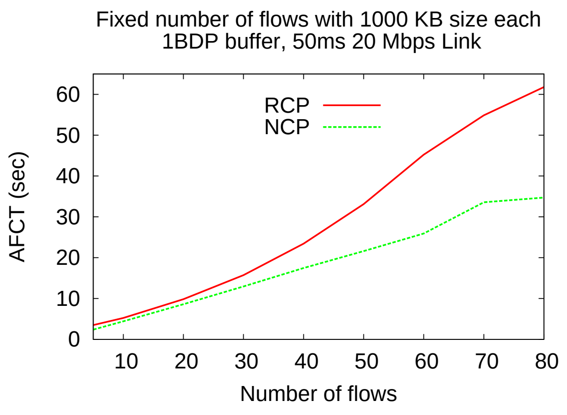

**Figure 3: AFCT versus number of flows**

## 8.1 Simulation Results

In this section we present results first for single bottleneck and then for a two bottleneck network.

### *8.1.1 Single Bottleneck Network*

Similar to previous work on RCP, we first use a simple topology which contains sources and a destinations connected by one single link. Even though RCP is sensitive to the choice of stability and control interval parameters, in this analysis we use the parameters which give the best RCP performance.

As can be seen from Figure 3 where a fixed number of flows with a fixed file size was used, the AFCT of NCP is much smaller than that of RCP.

Most of the experiments used to validate RCP in the literature were obtained using a non-congestion scenario with an average load of for instance 90%. However such a simulation scenario doesn't properly evaluate the performance of RCP. In fact as in a *Naive NCP* approach where we set the initial *cwnd* of every flow equal to the file size of the flow for the cases where the link on average is not fully utilized (similar to many RCP experiments in the literature), the network doesn't get congested on average as shown in figure 4. In this scenario a congestion control protocol is not even strictly needed as all flows can send all the packets they have in one round and retransmit some of their lost or delayed packets to get very small AFCT. As can be seen from the plot, even Naive NCP outperforms RCP.

However under a real congestion scenario, the performance of RCP is worse when compared with NCP as shown in the next experimental results. In these experiments Poisson flow arrivals where the file sizes are Pareto distributed are used. As shown in table 2, within a simulation time of 26.061 seconds only 17280 RCP flows finished due to the increasingly high file completion time (FCT) as shown in figure 5. On the other hand as can be seen from table 2, 66212 NCP flows finished during the same time.

We also compared the FCT of the 17280 RCP flows (all RCP flows) which finished against the first 17280 NCP flows which finished. As shown in figure 6 the FCT of NCP flows is much smaller than that of RCP. This small FCT helped

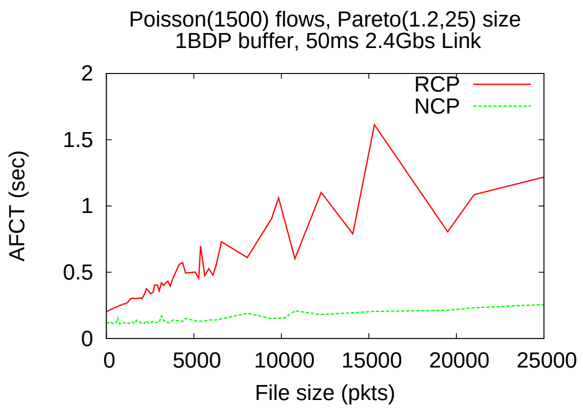

**Figure 4: AFCT of *Naive NCP* vs RCP**

**Table 2: NCP versus RCP under high load scenario: Poisson(8333.3), Pareto(1.2,30)**

| Protocol | Number of finished flows (in 26.061 sec) |
|---|---|
| RCP | 17280 |
| NCP | 66212 |

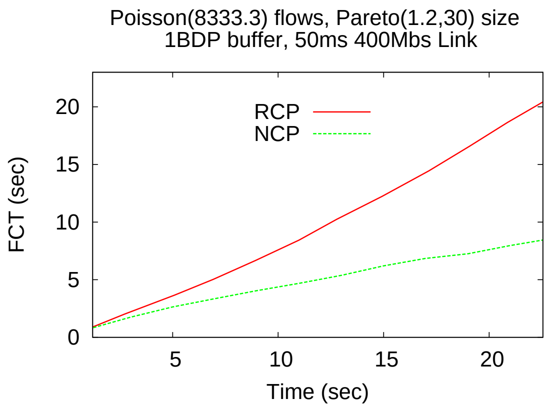

**Figure 5: FCT of flows versus simulation time (with 2 sec aggregation)**

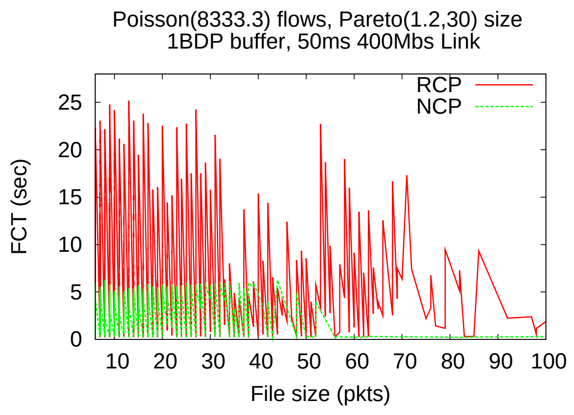

**Figure 6: FCT of finished RCP flows vs NCP flows**

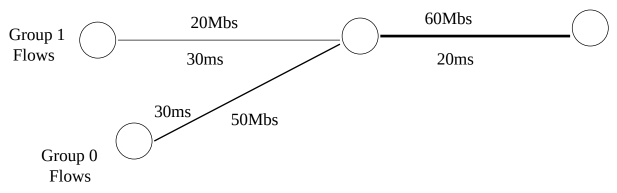

**Figure 7: Two botleneck network topology**

more NCP flows finish in a shorter time as shown in table 2.

### 8.1.2 *Multiple Bottleneck Network*

We have also compared the performance of NCP against RCP for a two bottleneck network topology as shown in figure 7.

As shown in figure 8, NCP gives lower FCT for the two groups of flows crossing two different bottleneck links as shown in the topology of figure 7.

## 9. SUMMARY

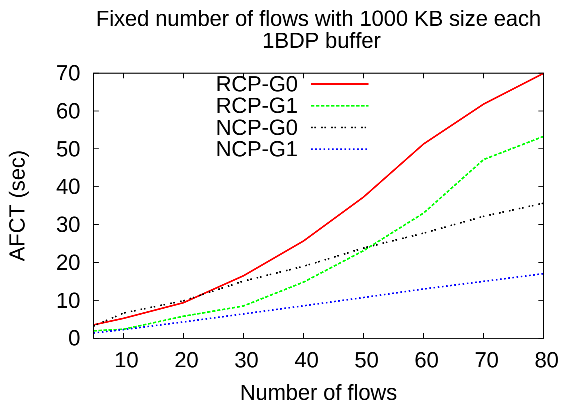

**Figure 8: FCT of Group 0 (G0) and Group 1 (G1) RCP and NCP flows**

In this paper we presented the design of a Network congestion Control Protocol (NCP). NCP uses a *fair rate metric* to determine *the rate at which flows send data*. We have shown how NCP can achieve a more efficient sharing (ES) scheme than the traditional processor sharing (PS).

Simulation results show that NCP can outperform the Rate Control Protocol (RCP), which is a well known congestion control protocol. We have also made initial stability analysis of NCP describing that NCP is stable for a wide range of parameter settings.

## 10. REFERENCES

[1] H. Balakrishnan, N. Dukkipati, N. McKeown, and C. Tomlin. Stability analysis of explicit congestion control protocols. *Communications Letters, IEEE*, 11(10):823–825, October 2007.

[2] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *IWQoS*, pages 271–285, 2005.

[3] D. Fesehaye. A Fast Congestion control Protocol (FCP) for Networks (the Internet). In *ITRE*, pages 131–135, 2006.

[4] S. Floyd. Highspeed TCP for large congestion windows, 2002.

[5] T. R. Henderson, E. Sahouria, S. McCanne, R. H. Katz, and Y. H. Katz. On Improving The Fairness Of TCP Congestion Avoidance. In *IEEE Globecomm conference*, pages 539–544, 1997.

[6] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, New York, NY, USA, 1988. ACM.

[7] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):89–102, 2002.

[8] L. Kleinrock and R. R. Muntz. Processor sharing queueing models of mixed scheduling disciplines for time shared system. *J. ACM*, 19:464–482, July 1972.

[9] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Netw.*, 5(3):336–350, 1997.

[10] Z. Lu and S. Zhang. Stability analysis of XCP congestion control systems. In *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*, pages 1–6, april 2009.

[11] S. McCanne and S. Floyd. NS-2. `http://www.isi.edu/nsnam/ns/`.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, March 2008.

[13] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1:344–357, June 1993.