A Network Congestion control Protocol (NCP)

Debessay Fesehaye, Klara Nahrstedt and Matthew Caesar
Department of Computer Science
UIUC, 201 N Goodwin Ave
Urbana, IL 61801-2302, USA
Email:{dkassa2,klara,caesar}@cs.uiuc.edu

Abstract

The transmission control protocol (TCP) which is the dominant congestion control protocol at the transport layer is proved to have many performance problems with the growth of the Internet. TCP for instance results in throughput degradation for high bandwidth delay product networks and is unfair for flows with high round trip delays. There have been many patches and modifications to TCP all of which inherit the problems of TCP in spite of some performance improvements.

On the other hand there are clean-slate design approaches of the Internet. The eXplicit Congestion control Protocol (XCP) and the Rate Control Protocol (RCP) are the prominent clean slate congestion control protocols. Nonetheless, the XCP protocol is also proved to have its own performance problems some of which are its unfairness to long flows (flows with high round trip delay), and many per-packet computations at the router. As shown in this paper RCP also makes gross approximation to its important component that it may only give the performance reports shown in the literature for specific choices of its parameter values and traffic patterns.

In this paper we present a new congestion control protocol called Network congestion Control Protocol (NCP). We show that NCP can outperform both TCP, XCP and RCP in terms of among other things fairness and file download times.

Keywords: *Congestion Control, clean-slate, fairness, download times.*

1 Introduction

Communication networks are at the core of every technological advancement as entities cannot perform a reasonable task with out communicating some information. At the core of *communication networks* research in turn is *the resource allocation* problem to avoid congestion (contention) as the entities compete for scarce (bottleneck) resources. So bottleneck resource allocation problems deal with *the congestion control* problem. This paper aims at addressing this problem when the bottleneck resource is *link capacity* of computer networks. The scheme can be adapted for other bottleneck resources.

The current widely used protocol to address the congestion problem in computer networks is the transmission control protocol (TCP) [10]. Even though there are various implementations and extensions of TCP, it generally involves the *slow start (SS)* and *congestion avoidance (CA)* algorithms. A TCP source begins with the *SS* algorithm where it sends two packets every time it receives an acknowledgement (ACK) of the previously sent packets until it reaches the slow start threshold *ssthresh*. This results in an exponential increase of the window w of packets the source sends every round trip time (RTT). When the the *ssthresh* is reached the TCP source starts the CA algorithm where it increases the window size by one every RTT and hence a linear window size increase. Different implementations of TCP use different approaches on how to (multiplicatively) decrease the window size w (how many packets to send) when a packet is lost. TCP assumes packets are lost if the ACK times out or if (triple) duplicate acknowledgements arrives. More details of TCP can be found in [10].

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. In particular TCP either under utilizes or over utilizes the network bandwidth resulting in a download time much longer than necessary. The performance limitations of TCP over high bandwidth-delay product networks has been reported in [13]. They showed that a random packet loss can result in a significant throughput degradation. The same paper also show that TCP is grossly unfair towards flows with higher round trip delays. On the other hand TCP is not fair for short-lived flows as shown in [9] as the bottleneck bandwidth is dominated by long-lived flows whose window size has grown so large. As has been extensibly reported in the literature [16] TCP is also not suitable for wireless networks. The main reason is that TCP assumes that all packet losses are due to network congestion while in the case of wireless networks it can be due to some wireless link errors which may correct themselves in the next round.

Both the user datagram protocol (UDP) and TCP are transport layer protocols. While TCP is a reliable protocol which makes sure that packets are received by the destination, UDP is unreliable protocol which just cares about the speed of the communication and gives no guarantee that packets are received by the destination. The unreliable nature of UDP can cause congestion collapse and there are new TCP-friendly proposals like the Datagram Congestion Control Protocol (DCCP) [12] to deal with these problems. DCCP is based on the TCP algorithms. There are many variants of and modifications to TCP an example of which is the HighSpeed TCP [8]. Nonetheless they all inherit the basic limitations of TCP in spite of some improvements over the original TCP.

On the other hand there are clean-slate design protocols like XCP [11] and RCP [4] to deal with limitations of TCP and avoid network congestion. The Network congestion Control Protocol (NCP) we present in this paper largely belongs to this category. We will discuss these protocols further in section 2 below.

To this end the main contributions of this report are as follows.

- It points out the limitations of XCP and RCP which are the well known existing clean slate congestion control protocols in addition to similar studies in the literature. It further gives exact derivations of conditions under which XCP and RCP perform or don't perform well.
- It presents NCP which is a noble congestion control protocol and proves that NCP can generalize both XCP and RCP.
- It presents an exact characterization of the average file download time of processor sharing (PS) systems.
- It provides one extension of NCP for scenarios where it is easy to count the number of active flows sharing a link (resource) and another extension which improves NCP fairness for scenarios where the variance of the RTTs of the flows is too high and when the number of active flows changes every round.
- The paper presents an easy proof of the feasibility of the well known early deadline first (EDF) and the Pfair scheduling algorithms using a simple PS and rate concept.
- The paper briefly describes possible NCP case studies for overlay networks (TEEVE applications), an active queue management approach (the core-stateless fair queueing) and scheduling.
- The paper also presents some NS2 simulation results and briefly describes a simple but accurate simulator which can be used to simulate different feedback and non-feedback based systems.

The rest of this paper is organized in such a way that we first discuss the related work in section2. We then present the formulation of NCP, how NCP works and how it achieves PS in sections 3, 4 and 5. In sections 6 and 7 we discuss some refinements and extensions of NCP. Following this in section 8 we show how NCP can generalize existing clean-slate design protocols and in section 9 how the simple rate based ideas used in NCP can be used with a scheduling algorithm. In section 10 we discuss some case studies where NCP can be used and discuss performance evaluation of NCP against XCP and NCP, the two well known clean-slate protocols in section 11. Finally we give a brief summary and description of the ongoing work in section 12.

2 Related Work

The major congestion control protocols which fall under the Clean-Slate Internet design category are the eXplicit Congestion control Protocol (XCP) and the Rate Control Protocol (RCP). Under XCP all flows increase their sending rates by the same amount if there is available bandwidth and they all decrease it if the network is loaded. This means that both long and short flows increase their sending rates at the same time making XCP unfair to short-lived flows as shown in section 2.1 below. Besides, XCP requires many per packet computations at the routers. The RCP scheme on the other hand tries to approximate processor sharing (PS) by making a rough estimation of the number of active flows at a router. This estimation is the main limitation of RCP as shown in section 2.2 below.

2.1 On the Performance of XCP

The fact that XCP is not fair to short flows (flows with small data to send) makes its average file completion (download) time (AFCT) much higher than TCP as shown in [5]. For example let's consider three short lived flows which just started with a congestion window size of 1 and need to send 50 packets each and one long lived flow which needs to send 500 packets and already has a window size of 60 packets. Without loss of generality let's assume that they all have the same round trip time (RTT). If the spare link capacity is 20 packets per RTT then XCP shares it equally among all four flows allowing each flow to increase its congestion window by 4 packets per RTT. This implies that the window size of the three short lived flows is now set to 5 packets per RTT. Hence it takes $50/5 = 10$ rounds (RTT) to download each of the short lived flows and hence a longer AFCT. But NCP and RCP attempt to reduce this by dividing the entire link capacity (say

80 packets/RTT) equally among all four flows. This implies that each flow sets (resets) its window size to 80/4 = 20 packets per RTT. This implies that each of the short lived flows (the majority) will have a file download time of about 2.5 rounds (RTT). We will discuss more about how the rate allocation scheme of XCP differs from that of NCP in section 8.2.

2.2 On the Performance of RCP

The rate update equation of the newly proposed rate control protocol (RCP) [4] for the Internet is given by

$$R(t) = R(t - d_0) + \frac{(\alpha(C - y(t)) - \beta\frac{q(t)}{d_0})}{N(t)} \tag{1}$$

where d_0 is a moving average of the RTTs measured across all packets, $R(t - d_0)$ is the last (previous) updated rate, C is the link capacity, $y(t)$ is the measured input traffic rate during the last update interval (d_0 in this case), $q(t)$ is the instantaneous queue size, $N(t)$ is the router's estimate of the number of ongoing flows (i.e. number of flows actively sending traffic) at time t and α, β are parameters chosen for stability and performance.

In RCP and the rate control protocol with acceleration control (RFC-AC) [6], the number of ongoing flows, $N(t)$ is estimated as

$$N(t) = \frac{C}{R(t - d_0)}. \tag{2}$$

But this is a heuristic estimate and is where the major limitation of RCP lies.

So RCP either over-estimate or under-estimate the allocated rate $R(t)$. When the initial value of $R(t-d_0)$ from which $N(t)$ is obtained is too small then $N(t)$ is too large . This in turn results in the router unnecessarily dividing the capacity into too many flows resulting in link under-utilization. Let's consider an initial rate of $R(t - d_0) = C/200$ whose corresponding $N(t) = 200$. If the link receives only 40 flows/sec for an RTT of 0.1 sec, we have an actual number of 4 flows. If the router allocates each of these flows only $C/200$, then the total arrival rate for the next round becomes $C/50$ which is 1/50 of the available link capacity.

On the other hand if the initial value of $R(t - d_0)$ is too large, then $N(t)$ becomes too small. As a result the router divides the capacity into fewer number of flows and hence over-estimates the rate allocation. This causes link over-utilization, more queuing delays and packet losses. In fact the simulation setup of RCP uses a huge buffer capacity (to avoid this).

For example let the initial sending rate $R(t-d_0) = C/4$. Then the corresponding $N(t) = 4$. If the flow arrival rate is 200 flows/sec for an RTT of 0.1 sec, the actual number of flows is 20. The router then tells each of these 20 flows to send at the rate of $R(t-d_0) = C/4$. If they all send at this rate then the total arrival rate $\Lambda = 20C/4 = 5C$. Hence the link receives 5 times more packets than it can handle.

We next explain why RCP seems to closely emulate processor sharing in the published literature.

2.3 Does RCP really closely emulate processor sharing?

In [4] and other similar works, RCP is reported to closely emulate processor sharing (PS). In the simulation setup used to evaluate the performance of RCP flows arrivals are Poisson and flow sizes are Pareto distributed. These distributions are reasonable for the performance evaluation of such congestion control protocols. However the link load is fixed in all simulation setups (to be less than 1). In reality the link load $\rho = \Lambda/C$ where Λ is the total packet arrival rate, highly depends on the flow arrival rate and on the way the protocol allocates rate $R(t)$ to the flows. Hence the load should not be fixed. In the simulation setup the authors also calculate the average flow arrival rate f_{rate} as a direct function of the load and the average flow size f_{size} which is also fixed as follows.

$$f_{rate} = \frac{\rho C}{f_{size}} = \frac{\Lambda}{f_{size}}. \tag{3}$$

By fixing the values the authors are making sure that on average there will be no overflow even if all flows send on average all packets ($f_{rate} \times f_{size}$) they have (all files) in one round. Hence if the average RTT is 0.1 sec then the average flow completion time (AFCT) is about 0.1 sec for the SYN/ACK to discover the rate allocation plus about 0.05 sec for the flow to be completely transmitted plus about 0.05 sec processing time which gives about 0.2 sec which is the average value shown in the RCP papers. A similar argument applies when the link load is a fixed number greater than 1.

Therefore such “convenient simulation” approach on average hides the over-shooting nature of RCP even if on average all files of the flows are sent in one round. Thus RCP doesn’t really closely emulate PS unlike what is shown in the RCP plots of [4] as it under or over estimates the number of active flows into which the link capacity has to be allocated. We believe that the performance of such congestion control protocols should be evaluated by considering realistic and different what if scenarios. In particular a congestion control protocol shouldn't only be evaluated under no congestion (0.9 total load). In section 8.1 we summarize the scenarios where RCP works and doesn't work well. Nonetheless NCP uses an exact derivation for the number of flows and avoids all limitations of RCP and XCP as discussed in the following sections.

3 The NCP Formulation

The NCP rate allocation can be formulated as follows. Let w_j be the current *cwnd* (congestion window) of a flow attached to the jth packet of the L_i packets which arrive to router i during the control interval d_0 and which is used to calculate the throughput $R(t)$ and the *cwnd* w'_j for the next round. Define the ***per packet throughput*** to be the number of packets a source sends per unit time at an arrival of each of the w_j ACKs of the (w_j) packets sent in the previous round.

The sum of the *per packet throughput* shouldn't exceed the link capacity minus the bandwidth needed to drain the queue within a round trip time (RTT) or within a control interval. That is

$$\sum_{j=1}^{L_i} \frac{R(t)}{w_j} = \alpha C - \beta \frac{q(t)}{d_0}. \tag{4}$$

This implies that

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d_0}}{\sum_{j=1}^{L_i} (1/w_j)}. \tag{5}$$

By using the estimation $w_j = d_0 R(t - d_0)$ in Equation 5 the NCP rate can be given by

$$R(t) = \frac{(\alpha C - \beta \frac{q(t)}{d_0}) R(t - d_0)}{\Lambda_i} \tag{6}$$

where $\Lambda_i = L_i/d_0$ is total packet arrival rate to router i. This simplified version of NCP is called NCP-S in this paper needs less work at the routers.

The NCP rate can also be derived using the fact that the total number of packets sent to a router (link) shouldn't exceed the bandwidth-delay product minus the queue size at the router. Hence if $R_j = w_j/RTT_j$ denotes the rate attached to the jth of the L_i packets which arrive to the router,

$$\sum_{j=1}^{L_i} \frac{R(t)}{R_j} = \alpha C d_0 - \beta q(t). \tag{7}$$

This implies that

$$R(t) = \frac{\alpha C d_0 - \beta q(t)}{\sum_{j=1}^{L_i}(1/R_j)}. \tag{8}$$

4 How NCP Works

- First each router in the network calculates $R(t)$ every control interval.
- A source sends a packet j with its desired rate R_j.
- Each router in the path of the flow checks if $R(t) < R_j$ in which case it overwrites R_j and forwards it unchanged otherwise.
- The destination then copies the R_j in the data packet to the ACK packet.
- The source sets its current window size $w'_j = R_j RTT_j$ upon receipt of the ACK packet.
- Each router updates its $R(t)$ value every control interval.

The routers also need the RTT_j which can be obtained by making small modification to the TCP *time stamp option* (see RFC1323). The modification is that the two four-byte time stamp fields (*TSval* and *TSecr*) should contain the previous and current time stamp values of the sender from which any router in the path can get the round trip time of the packet passing through it.

5 NCP Achieves PS

If we denote the total number of concurrent flows at router i with N_i, rearranging the headers of the packets which arrive to the router during a control interval, Equation 5 can be written as

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d_0}}{\underbrace{\frac{1}{w_1} + \frac{1}{w_1} + \cdots + \frac{1}{w_1}}_{w_1} + \underbrace{\frac{1}{w_2} + \frac{1}{w_2} + \cdots + \frac{1}{w_2}}_{w_2} + \cdots + \underbrace{\frac{1}{w_{N_i}} + \frac{1}{w_{N_i}} + \cdots + \frac{1}{w_{N_i}}}_{w_{N_i}}} \tag{9}$$

which is the same as

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d_0}}{N_i} \tag{10}$$

which in turn is the Processor Sharing (PS) rate. Hence NCP achieves processor sharing without having to count the exact number of concurrent flows at a router.

6 Extending (Refining) NCP: On the number of active flows

There are a couple of schemes [3] which try to count the number of active flows in a link by classifying packets. Apart from the extra overhead to classify packets and count the number of active flows, such schemes may not be as good as the NCP scheme in estimating the fair share $R(t)$. This is because such counting schemes cannot tell the fraction of each flow bottlenecked somewhere else in the network. This may cause a router to give more share to the flows bottlenecked elsewhere at the expense of the other flows bottlenecked at the allocating router. This can be clarified as follows.

If R_j denotes the rate of the bottleneck link so far which is the minimum of the rates of the routers crossed thus far, $R(t-d) = R$ denotes the rate at the current router calculated in the previous interval and $w = dR(t-d)$ is its corresponding window size, $R(t)$ which is the rate for the next interval can now be calculated as

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d}}{N - \sum_j^n (\frac{1}{w_j} - \frac{1}{w})} = \frac{\alpha C - \beta q(t)}{dN - \sum_j^n (\frac{1}{R_j} - \frac{1}{R})} \tag{11}$$

whenever $R_j < R$ where N is obtained by counting flows and n is the number of packets of the flows bottlenecked in the preceding routers. This can be given as

$$R(t) = \frac{\alpha C d - \beta q(t)}{dN + \frac{n}{R} - \sum_j^n \frac{1}{R_j}} \tag{12}$$

Of course n can be 0 in which case

$$R = \frac{\alpha C - \beta \frac{q(t)}{d}}{N}. \tag{13}$$

This additional refinement gives NCP an additional advantage over RCP, XCP and even processor sharing (PS).

7 Refinement of NCP: On the fairness among flows in a fast changing network

NCP like other explicit congestion control protocols such as XCP and RCP uses a fixed average control interval d to update the rate it allocates to the different flows. So if the variance of the round trip times (RTT) of the flows is too big and if the number of active flows changes every average RTT (d), then the rate allocation may not be fair for or against flows of short or long RTT.

For example if flow k has an RTT of $RTT_k > d$ then we have the following cases.

Case 1: $R_k \leq R(t)$
In this case flow k sends at $R(t) - R_k$ less than the actual allocation for $RTT_k - d$ time units (sec) as flow k updates its sending rate only after RTT_k sec. So flow k needs to be compensated for sending at a lower rate for $RTT_k - d$ time units to achieve fairness. Therefore

$$R_k = R(t) + \frac{(R(t) - R_k))(RTT_k - d)}{RTT_k}. \tag{14}$$

Case 2: $R_k \geq R(t)$
On this other case, flow k sends at $R(k) - R(t)$ more than the actual allocation for $RTT_k - d$ time units (sec) at the expense of other flows. So flow k should be penalized for sending at a higher rate for $RTT_k - d$ time units to be fair to the other flows. Hence

$$R_k = R(t) - \frac{(R_k - R(t))(RTT_k - d)}{RTT_k}. \tag{15}$$

If $RTT_k \leq d$ then NCP doesn't need any refinement as the flow always discovers the latest allocation.

8 NCP as a generalization of existing explicit congestion control schemes

In this section we present how XCP and RCP can be generalized by NCP. We also discuss situations where XCP and RCP perform or don't perform well.

8.1 General Cases when RCP works well: Derivation using NCP

Putting Equation 1 into Equation 7

$$N(t) = \frac{\left(\alpha C - \alpha y(t) - \beta \frac{q(t)}{d_0}\right) \sum_{j=1}^{L_i} (1/w_j)}{\alpha C - \beta \frac{q(t)}{d_0} - R(t-d_0) \sum_{j=1}^{L_i} (1/w_j)}. \tag{16}$$

Therefore the specific scenario where RCP works well is when the approximate value of $N(t)$ which is $C/R(t-d_0)$ equals the exact value given by Equation 16 above. That is when

$$\frac{C}{R(t-d_0)} = \frac{\left(\alpha C - \alpha y(t) - \beta \frac{q(t)}{d_0}\right) \sum_{j=1}^{L_i} (1/w_j)}{\alpha C - \beta \frac{q(t)}{d_0} - R(t-d_0) \sum_{j=1}^{L_i} (1/w_j)}. \tag{17}$$

This implies that

$$\alpha C^2 - \left(\beta \frac{q(t)}{d_0} + (1+\alpha) y(t)\right) C + y(t) \left(\alpha y(t) + \beta \frac{q(t)}{d_0}\right) = 0. \tag{18}$$

By solving this quadratic equation for different values of the constants we can see the specific scenarios where RCP works well.

For instance setting $q(t) = 0.0$, $\alpha = 0.1$, $\beta = 1.0$ and solving the quadratic equation using Maple we can see that RCP works well if $y(t) = 10.908C$ or $y(t) = 0.092C$. When $\alpha = 0.1$, $\beta = 1.0$, $q(t)/d_0 = 5$, $C = 10$ we get $y(t) = 25.62$.

For all values which do not satisfy Equation 18 RCP doesn't perform well by either causing delays and packet losses or by under-utilizing the links or by being so slow to converge to stability.

8.2 Deriving XCP from NCP

The rate allocation scheme of an XCP-like algorithm can be derived from the RCP ideas as follows. The main idea of XCP is to divide the spare bandwidth $S = C - \Lambda - q(t)/d_0$ among the active flows where Λ is the total packet arrival rate and the other variables are as defined above. If we denote the spare bandwidth share of each flow as ΔR then the sum of the per packet share of each flow should not exceed the total spare bandwidth S. Hence

$$\sum_{j}^{L} \frac{\Delta R}{R_j} = d_0 S. \tag{19}$$

This implies that

$$\Delta R = \frac{d_0 S}{\sum_j^L \frac{1}{R_j}}. \tag{20}$$

Hence a flow with a current sending rate of R_i sets its new sending rate R_i^{new} to

$$\begin{aligned} R_i^{new} &= R_i + \Delta R \\ &= R_i - \frac{d_0 \Lambda}{\sum_j^L \frac{1}{R_j}} + \frac{d_0 C - q(t)}{\sum_j^L \frac{1}{R_j}} \\ &= R_i - \frac{d_0 \Lambda}{\sum_j^L \frac{1}{R_j}} + R(t)^{ncp} \end{aligned} \tag{21}$$

where $R(t)^{ncp}$ is the rate allocation of NCP. From the above derivation it can be seen that XCP can behave like NCP if the rate at which flows send packets R_j is the same. If this value is known and the same for all flows then NCP, XCP and RCP all have the same performance.

The above representation of NCP (XCP) can now be modified to achieve different objectives. For instance one can multiply Equation 21 with R_j/R if it is needed to keep the flow sending rate proportional to its current rate or with R/R_j if one wants to adjust the sending rates to the equal share.

So from the analysis in the previous sections we can see that both XCP and RCP can be subsets of NCP. So NCP can be thought of as the generalization of such explicit (congestion) control protocols.

9 NCP and Scheduling

Here we first give a simple proof to the famous feasibility theorem by Liu and Layland [14] of dynamic deadline driven scheduling algorithm using the concept of rate as used in NCP. We then show how such rate approach can easily derive and generalize the Pfair Scheduling algorithm [1, 2] for multiprocessor scheduling.

9.1 Proving the feasibility of EDF

The EDF theorem states that for a given set of m tasks, the deadline driven scheduling algorithm is feasible if and only if

$$\sum_{i}^{m}(c_i/T_i) \leq 1 \tag{22}$$

where the c_i and T_i are the worst case execution time and period (deadline) of task i respectively.

The authors took the LCM of the periods to prove the theorem. We use the rate concept as used in NCP as follows. If processor capacity is C then number of instructions performed during c_i is c_iC. Hence the rate at which instructions corresponding to task i are performed is $R_i = c_iC/T_i$. But the sum of these rates shouldn't exceed the total processor capacity. Hence

$$\sum_{i}^{m} R_i \leq C \tag{23}$$

which implies that

$$\sum_{i}^{m}(c_i/T_i) \leq 1. \tag{24}$$

This is to say that the necessary and sufficient condition for such feasible scheduling is that the total demand should be less or equal to the supply.

9.2 Generalizing the Pfair Scheduling Algorithm

The scheduling algorithms for a single processor can not be directly used for multiple processors. One of the reasons is that when some of the tasks are scheduled in some of the processors, the remaining capacity of each of the processors may not be enough for any of the remaining tasks. This will require that the remaining tasks be divided into smaller subtasks which can fit into the remaining capacities of the processors. Scheduling algorithms like the Pfair (Proportional fair) [2] algorithms assume this kind of dividing tasks into subtasks. Most of the studies on Pfair scheduling assume identical processors and use a rather lengthy proof. Here we present a generalization of such algorithms and give a simple derivation.

Tasks $t_j, 1 <= j <= n$ with the worst case execution time (WCET) c_j (each derived using a processor of capacity C_j) and a deadline (period) P_j are schedulable in m processors of capacities $C_i, 1 <= i <= m$ if the total demand is less or equal to the total supply.

That is if

$$\sum_{j}^{n} C_j c_j / T_j \leq \sum_{i}^{m} C_i$$

where $C_j c_j / T_j$ is the *rate* at which task j is sending instructions to the processor.

Now we can have many variations of the above general formulation. For instance if all the processors are identical then the schedulability condition is

$$\sum_{j=1}^{n} c_j / T_j <= m, \qquad (25)$$

which is the case of Pfair.

In the next section we will show how NCP can be used as a scheduling scheme.

10 NCP Case Studies

Here we present short description of some case studies where NCP can be used.

10.1 NCP for Overlay Networks (TEEVE applications)

NCP as a congestion control protocol can be easily deployed in overlay networks such as the TEEVE [17] using computers with software routers. Each software router in the overlay network estimates its corresponding inbound and outbound link capacities, receives NCP frames (packets) and calculates $R(t)$. Each software router then adjusts the sending rate of the sources attached to it based on the feedback it receives from the corresponding ACK packets. Here each packet can be made to carry the interval $\Delta t = 1/R_j$ after which it is sent following the previous packet sent by the same source. This can then be easily used in NCP. QoS parameters can also be assigned to each frame where NCP allocates different bandwidth accordingly. We expect NCP to outperform any of the schemes which are currently deployed in overlay networks.

10.2 NCP and the CSFQ

NCP can also be implemented as a simple core-stateless fair queuing (CSFQ) [15]. Here NCP can use the packet inter-arrival time $\Delta t = 1/R_j$ to the ingress router used in the CSFQ algorithm. This assumes that the packet inter-arrival time Δt is the same as the interval between two packet sends by the source. If the link which connects the source to the ingress router is congested, then the switch (computer) cannot send any packet. NCP can still be implemented as CSFQ without this simple assumption. We hope that this scheme will achieve more simplicity and accuracy over CSFQ, future work and analysis should show how such implementation of NCP compares with CSFQ though.

10.3 NCP as a Scheduling Algorithm

If we consider each packet of a flow which carries the corresponding R_j and RTT_j as a task, then all these tasks are schedulable if Equation 23 holds. This is equivalent to saying if $R_j \leq R(t)$. This can as well be seen as a simple admission control protocol which admits packets if their R_j is less or equal to the what the link can accommodate which is $R(t)$. Here NCP can also be modified to ensure that a task with a priority of p_j is schedulable if $R_j \leq p_j R(t)$.

11 Performance Evaluation

In this section we first present an exact characterization of AFCT of PS systems which many congestion control protocols try to emulate and we then discuss numerical results which compare the performance of NCP against XCP and RCP.

11.1 Exact characterization of the Average File Completion Time (AFCT) of PS systems

A processor sharing system with processing capacity C allocates $R(t) = C/N$ of its capacity to N sources (flows). For a flow arrival rate of f during a control interval d the number $N = fd$ using Little's law (independent of the probability distribution). Hence

$$R(t) = \frac{C}{fd}. \tag{26}$$

Using Little's law a file with an average size of $\bar{F}$ needs $\bar{F}/R(t)$ time units (sec) to download (complete). With an overhead of about $1.5RTT$ to setup and tear down a connection,

$$AFCT_{PS} = 1.5RTT + \frac{\bar{F}fd}{C}. \tag{27}$$

11.1.1 Total link load (Link Pressure) vs link load per round

The average link load ρ used in the literature is load (demand) a link can accommodate at one time (in one round) and is usually assumed to be less than one otherwise the queue length grows to infinity. In this paper we define the *average total load* or with sith slight name abuse *link pressure* (ρ_T) as the ratio of the total average demand Λ_T to the to the total supply C. Here it should be noted that a PS system may fulfill the total demand in one round d in which case $\rho = \rho_T \leq 1$ or in many rounds n_r when the total demand is finite (flows of finite size for a finite length of time). Here excluding the connection setup and tear down time a flow completes after n_rRTT. The average total load (link pressure) $\rho_T = n_r\rho$.

So for an average total load of $\rho_T = \Lambda_T/C$ and average file size of $\bar{F}$, the flow arrival rate $f = \rho_T C/\bar{F} = \Lambda_T/\bar{F}$. Hence

$$AFCT_{PS} = 1.5RTT + d\rho_T \approx RTT(1.5 + \rho_T). \tag{28}$$

For example when $RTT = 0.1sec$ and $\rho_T = 0.9$ as is the case in RCP papers, the AFCT of PS and hence NCP is $AFCT_{PS} = AFCT_{NCP} = 0.24$ sec as can also be seen from Figure 1.

11.2 Numerical Results

In this section we compare the performance of NCP (which achieves PS as shown above) with that of XCP and RCP using the ns2 simulation package. The performance of RCP highly depends on the choice of the many parameters it uses. As shown in Figure 1 RCP seems to outperform all but NCP when the flow arrival is exponential with rate

$$f = \frac{\rho C}{\bar{F}} \tag{29}$$

where ρ and $\bar{F}$ are the link load and mean flow size respectively.

Multiplying both sides of Equation 29 with the average flow completion time $AFCT$ we get

$$N_{flow} = \rho C/R_{flow}$$

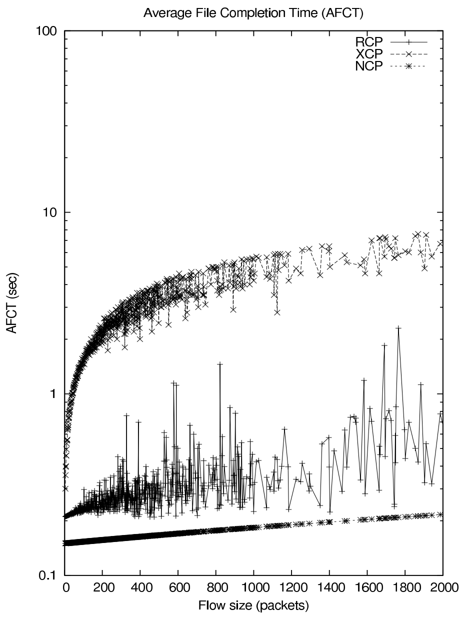

Figure 1: Average File Completion Time with smaller flow arrival rate

where N_{flow} and R_{flow} are the average total number of flows and average flow sending rates respectively. So with this initial values and simulation setup which give the almost exact estimate of the number of active flows, it is not surprising that RCP gives results close to PS. This however is not always the case with real networks. For example if the flow arrival rate is exponential with rate $f = 2\rho C/\bar{F}$ the performance of RCP is worse than XCP and TCP as shown in Figure 2.

We further validated these congestion control protocols using our own simpler but efficient *SimpSim* simulator which is briefly described below.

11.2.1 The *SimpSim* Simulator

In the *SimpSim* simulator there are source and network models which continuously exchange data. The *SimpSim* simulator is specially suitable for reliable feedback protocols such as TCP, XCP, RCP and NCP where the sources send packets to the network and the network controls their sending rate in different ways. This feedback behavior of reliable protocols is better modeled with the fixed point theory [7] which our *SimpSim* is based. The *SimpSim* simulator can also be easily used with non-reliable protocols such

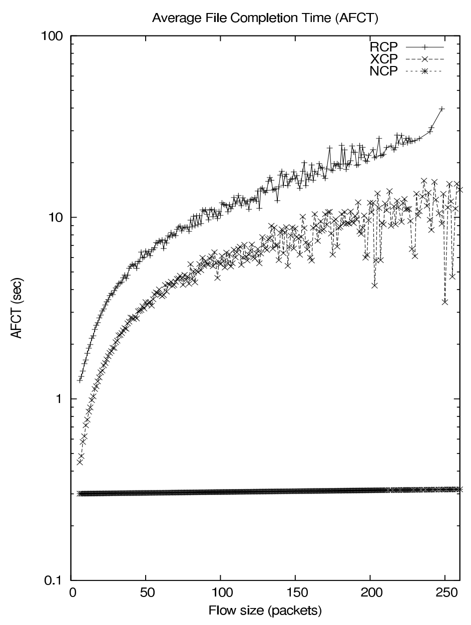

Figure 2: Average File Completion Time with larger flow arrival rate

as UDP. Here the source model of the simulator sends data at the sources rate and the network model measures (calculates) the performance metric without sending a feedback.

The *SimpSim* uses the rich field of queueing theory and accurate analytical models derived from the protocols to get the performance metrics. It can be shown that *SimpSim* can scale to bigger networks than the *ns2* simulator.

As can be seen from Figures 3, 4, 5 and 6 RCP doesn't always outperform TCP. The simplified version of NCP which we call NCP-S outperforms them all and NCP outperforms both TCP and RCP. All AFCT results are for file sizes ranging from 200 to 800 packets. The link capacity is in packets/sec, the buffer size is in packets and the AFCT (avgCompletionTime) is in seconds. The link capacity is in packets/sec, the buffer size is in packets and the AFCT (avgCompletionTIme) is in seconds.

```
     C           buffer    nIterToBeStable avgCompletionTime
-------------------------------------------------------------
  200.000000    20.000000     88             18.355020
  300.000000    30.000000     57             12.781763
  400.000000    40.000000     38             8.212026
  500.000000    50.000000     28             5.696982
  600.000000    60.000000     23             4.591052
  700.000000    70.000000     19             3.617752
  800.000000    80.000000     17             2.935318
  900.000000    90.000000     15             2.712573
  1000.000000   100.000000    15             2.650614
```

Figure 3: TCP AFCT

```
     C           buffer    nIterToBeStable avgCompletionTime
 -----------------------------------------------------------
  200.000000    20.000000    92             18.050947
  300.000000    30.000000    73             13.910848
  400.000000    40.000000    62             11.606824
  500.000000    50.000000    54             10.050722
  600.000000    60.000000    49             9.020102
  700.000000    70.000000    45             8.125041
  800.000000    80.000000    41             7.424306
  900.000000    90.000000    38             6.789594
  1000.000000   100.000000   36             6.345465
```

Figure 4: RCP AFCT

```
   C          buffer     nIterToBeStable avgCompletionTime
-----------------------------------------------------------
 200.000000    20.000000    42             8.917652
 300.000000    30.000000    28             5.914731
 400.000000    40.000000    23             4.595455
 500.000000    50.000000    18             3.597167
 600.000000    60.000000    15             2.975374
 700.000000    70.000000    14             2.686840
 800.000000    80.000000    12             2.276415
 900.000000    90.000000    11             2.075967
 1000.000000   100.000000   10             2.007141
```

Figure 5: NCP AFCT

```
    C          buffer    nIterToBeStable avgCompletionTime
-----------------------------------------------------------
 200.000000    20.000000   42             8.892529
 300.000000    30.000000   28             5.894903
 400.000000    40.000000   22             4.455842
 500.000000    50.000000   18             3.449792
 600.000000    60.000000   15             2.825743
 700.000000    70.000000   14             2.532917
 800.000000    80.000000   12             2.245854
 900.000000    90.000000   11             2.046303
 1000.000000   100.000000  10             1.862041
```

Figure 6: NCP-S AFCT

12 Summary and Ongoing Work

In this paper we showed the limitations of existing clean-slate congestion control protocols and presented a new congestion control protocol called NCP. Through detailed mathematical analysis and some simulation results we have shown that NCP can outperform existing congestion control protocols. We gave brief descriptions of some case studies where NCP can be used.

We are currently working on more detailed simulation results and router implementation of the NCP protocol using the click software router. We plan to conduct extensive implementation tests of NCP in the TEEVE overlay network and other experimental networks.

References

[1] J. H. Anderson and A. Srinivasan. Pfair scheduling: beyond periodic task systems. In *RTCSA '00: Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, page 297, Washington, DC, USA, 2000. IEEE Computer Society.

[2] S. K. Baruah. Fairness in periodic real-time scheduling. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium*, page 200, Washington, DC, USA, 1995. IEEE Computer Society.

[3] Q. Dong, S. Banerjee, J. Wang, and D. Agrawal. Wire speed packet classification without tcams: a few more registers (and a bit of logic) are enough. *SIGMETRICS Perform. Eval. Rev.*, 35(1):253–264, 2007.

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

[5] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, 2006.

[6] N. Dukkipati, N. McKeown, and A. Fraser. Rcp-ac: Congestion control to make flows complete quickly in any environment. In *INFOCOM 2006 Proceedings. 25th IEEE International Conference on Computer Communications.*, pages 1–5. IEEE Computer Society, 2006.

[7] D. Fesehaye. Emulating TCP (a reliable internet protocol) using a fixed point algorithm. *Local Computer Networks, Annual IEEE Conference on*, 0:159–165, 2006.

[8] S. Floyd. Highspeed tcp for large congestion windows, 2002.

[9] 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.

[10] 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.

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

[12] E. Kohler, M. Handley, and S. Floyd. Designing dccp: congestion control without reliability. *SIGCOMM Comput. Commun. Rev.*, 36(4):27–38, 2006.

[13] 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.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[15] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Trans. Netw.*, 11(1):33–46, 2003.

[16] S. O. This, B. M. Kojo, and N. Vaidya. End-to-end performance implications of links with errors. In *BCP 50, RFC 3155*, 2001.

[17] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S.-h. Jung, and R. Bajscy. Teeve: The next generation architecture for tele-immersive environment. In *ISM '05: Proceedings of the Seventh IEEE International Symposium on Multimedia*, pages 112–119, Washington, DC, USA, 2005. IEEE Computer Society.