Efficient Cross-layer Routing & Congestion Control Architectures for Distributed Systems

PhD. Thesis Proposal

Debessay Fesehaye
Department of Computer Science
University of Illinois at Urbana-Champaign
dkassa2@illinois.edu

January 14, 2012

Abstract

In this proposal we first describe some major research problems in distributed systems. We broadly classify these research challenges into four categories. Each of these challenges are in the areas of *congestion control*, *routing*, *large scale distributed systems (cloud computing)* and *Distributed of Denial of Service (DDoS) attacks*. We then discuss existing techniques to deal with these challenges and the shortcomings of the techniques. Finally we propose our comprehensive *cross-layer congestion control and routing* approach along with *other schemes* to deal with all these challenges.

The design of the *Network Control Protocol (NCP)* to deal with the congestion control and routing problems is discussed in this proposal. The design of *Efficient Distributed File System (EDFS)* which uses the NCP concept along with other schemes to deal with the third challenge is also explained in this proposal.

Simulation results presented in this proposal show that the congestion control component of NCP outperforms the rate control protocol (RCP) which is a well known congestion control protocol. More simulation results also show that the congestion control component of EDFS gives higher performance than the congestion control component of the Google File System (GFS) which is the transmission control protocol (TCP).

As described in the research time-line section of this proposal, the next steps of this PhD research are mainly on the *routing (cross-layer)* components of NCP and EDFS among other things.

Contents

Chapter 1

Introduction and Related Work

Various types of distributed systems such as the Internet and other large and small scale networks have been shaping the way we live. The *Internet* which is the biggest distributed system and other types of distributed systems such as cloud networks have been growing so fast and complex. Our reliance on them has also been growing as fast. *Cloud computing* is the new emerging large scale distributed system. Cloud computing offers scalable storage and processing capacity. It has been used for various applications.

The application scope of cloud computing is even extended by introducing *cloudlets* which are decentralized and widely-dispersed Internet infrastructure whose compute cycles and storage resources can be leveraged by nearby mobile computers. The cloudlets are connected to the main clouds and the mobile device functions as a thin client, with all significant computation occurring in the nearby cloudlet [45, 47].

The growth in size and complexity of networks also resulted in a distributed system which is against other distributed systems. Malicious nodes from different locations can also form a large scale distributed system to target a bottleneck link (resource) and block all legitimate flows passing through it using *Denial of Service (DoS) and Distributed DoS (DDoS)* attacks. Such an attack [4, 25, 39] can cause significant economic and other damages to those using the target network.

The fast growth and complexity of these distributed systems brings along many challenges. In this chapter we will first present the major challenges we are addressing in this work in section 1.1. We will then discuss the existing schemes to address these challenges in section 1.2. In section 1.3 we introduce our proposed solutions to deal with each of the challenges. Finally, in section 1.4 we explain how the rest of this proposal is organized.

1.1 Major Challenges of Distributed Systems Addressed

The major challenges in distributed systems we are addressing in this work are as follows.

1. One of the major challenges of distributed systems is finding *the rate* at which distributed entities transmit data packets to each other. This is a *a congestion control* problem. For instance if the entities send packets at a higher rate, a bottleneck link they share somewhere along their path to their destination can be congested dropping many of the packets and causing a very high data transfer delay.

2. The second challenge is finding *the path (route)* of the data transmission from a source entity to the destination entity. This is a *routing* problem. For instance, if all sources follow a fixed path passing through a single bottleneck link, the link gets congested decreasing the rate at which the entities send data. This results in a decrease in throughput and increase in file download (transfer) time.

3. The use of large scale distributed cloud computing systems has become very common. The third challenge specific to these new kind of network structures is then to know what the most efficient way *to select servers* is, what the *best rate* to transmit data is, *where* in the cloud the best location to store data is and along which *path*. If the cloud services are used by mobile devices connected to cloudlets, then one also needs to know *how many cloudlets* are needed to give full coverage to a given area.

In these kind of distributed systems such as data center (cloud) networks there may be resource bottlenecks other than link capacities. For instance a server may be too busy with some CPU intensive computation to receive new data even though its up-link and down-link have enough capacity. The server may also not have enough storage capacity even though its CPU, its up-link and down-link have enough capacities left. So another question which we briefly address in this work aims to find a *multiple resource bottleneck rate allocation scheme*. Such rate allocation scheme aims to take multiple bottleneck resources such as link, CPU processing, and storage capacities into account.

4. The fourth challenge is common to general networks which occurs when many malicious nodes at different locations target an important link and break it using Denial of Service (DoS) and Distributed DoS attacks. So the challenge here is to design *the best architecture to deal with DoS and DDoS attacks*.

In the following sections we will discuss current approaches to deal with these challenges and their shortcomings. We will then introduce our solution to deal with all four challenges.

1.2 Existing Schemes and their Shortcomings

A lot of research has been done to address the challenges mentioned in the previous section. In this section we present the major existing research in each of the above challenging areas.

1.2.1 Existing Congestion Control Schemes

In this section we present existing major congestion control and routing schemes.

Deployed Congestion Control Protocol

The majority of network traffic uses the transmission control protocol (TCP) [33] 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. 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 [40]. They showed that a random packet loss can result in a significant throughput degradation. The same paper also shows that TCP is grossly unfair towards flows with higher round trip delays. TCP is also not fair for short-lived flows as shown in [32] 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 [8], 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.

Other Congestion Control Schemes

There have been numerous research efforts to deal with the weaknesses of the deployed congestion control (TCP). While many of them rely on modifications to the existing schemes such as TCP, others have proposed clean-slate approaches to congestion control. The current modifications to existing systems vastly inherit the main problems of the original systems and have not properly addressed the main challenges. The Datagram Congestion Control Protocol (DCCP) [38] which is primarily designed to replace the User Datagram Protocol (UDP) whose unreliable nature can cause congestion collapse is for instance based on the TCP algorithm. There are also many other variants of and modifications to TCP, an example of which is the HighSpeed TCP [26]. Nonetheless they all inherit the basic limitations of TCP in spite of some improvements over the original TCP as they mainly rely on packet loss and packet delay as congestion signals.

The major clean slate congestion control protocols such as the eXplicit Congestion control Protocol (XCP) [37] and the Rate Control Protocol (RCP) [10] also have their own major performance and implementation challenges. For instance among other things XCP is not fair to short flows which are the majority of Internet

flows resulting in higher average file completion time (AFCT). RCP tries to solve this problem of XCP. However, the way RCP estimates the number of active flows which it needs to obtain the rate at which flows should send packets is a major drawback. It can result in under or over estimation of the number of flows resulting in under or over utilization of bottleneck link capacity. This in turn results in very high queue length and packet drops which translates into a high AFCT. The fact that RCP is sensitive to the values of its many parameters and that there is no rule on how to set them is another major draw back of RCP.

1.2.2 Existing Routing (Traffic Engineering) Schemes

In this section we first discuss the currently deployed routing scheme. We then visit some other traffic engineering schemes.

Deployed Routing Scheme

The Open Shortest Path First (OSPF) [44] is the currently deployed routing protocol to find a path from one node in a local autonomous network to another node (entity). It is the most commonly used intra-domain Internet routing protocol. However OSPF as well is finding it increasingly difficult to cope with the growth in size and complexity of distributed systems. One of the main problems with the existing such Shortest Path Routing (SPR) is the simple heuristic routing metric (link weight) they use. The routing metric used doesn't properly take into account the latest status of the network.

Lack of efficient routing and congestion control protocols and algorithms has been forcing owners of big distributed systems to over-provision their resources (networks). Unfortunately apart from the cost of upgrading the network (distributed system), the Moores Law-like technology over-provisioning trend with the growth of for instance the Internet is not sufficient to contain congestion as shown by Akella et. al [2]. This is because, as the authors pointed out, the maximum congestion in the Internet scales poorly with the growing size of the Internet graph. Akella et.al have further shown that the famous SPR which is the routing protocol in the Internet today can be worse than the Border Gateway Protocol (BGP) which is a Policy Routing. This surprising result is not because trial and error is better than a scientific approach. It only exposes with a counter example the weaknesses of the existing SPR protocol demanding for a more clever and comprehensive scientific approach, something we hope to deliver in this work.

Other Routing or Traffic Engineering (TE) Schemes

A traffic engineering technique based on some ideas of XCP, (TeXCP) [34] was also proposed to address some routing issues. But TeXCP also inherits some of the unfairness properties of XCP. Besides, an Internet Service Provider (ISP) needs to configure each TeXCP ingress-egress agent with a set of K-shortest paths it can use to deliver its ingress-egress (IE) traffic. These paths are then pinned using a standard protocol like Multi-protocol Label Switching (MPLS) [46]. The shortest path here doesn't take congestion into account. It merely uses propagation delay as a link metric. So essentially if all these links are congested TeXCP will stick with them even if there are other less congested paths. TeXCP also needs additional probe packets to discover the utilization in each path. A traffic engineering (TE) technique for MPLS networks [35] was also proposed. But it is based on the notion that the number of flows (LSP requests) through a link can be known and is hence difficult to apply for non MPLS networks. Wang et.al [50] proposed an edge-based TE for OSPF networks. The scheme called a *k-set* TE method, partitions traffic into uneven k traffic sets at the edge of a network. For each traffic class (set), the k-set approach uses residual bandwidth (spare) capacity as a link weight and relies on a heuristic to solve a mathematical programming formulation. Such spare bandwidth link weight doesn't take into account the number of active flows in each link. For instance two links with the same spare bandwidth but different number of active flows are treated the same way. But this is not always true as the link with more flows is highly likely to be more loaded with time.

1.2.3 Existing Cloud Computing Architectures

The emergence of new kinds of distributed systems such as cloud computing also means the emergence of new challenges. The Google File System (GFS) [27] and its derivatives such as the Hadoop File System (HDFS) [48] which are the most commonly used distributed file systems for cloud computing are designed to meet

these challenges. These cloud systems use a single name node server (NNS) to keep metadata, replication and location information of all data chunks stored in the cloud. With growth of cloud network demand, the single name node can become a bottleneck resource and a single point of failure. Besides, such cloud systems use TCP with all its weaknesses to transfer data from one node to another node. Besides, they do not have an efficient scheme to decide where to store data, where to retrieve it from and how to route it. There has been a modification of TCP for data center networks (DCTCP) [3] an effort to improve on TCP for data center networks. However, DCTCP also inherits the problems of TCP that it depends on packet loss and packet markings as congestion signal. Besides DCTCP makes an unrealistic assumption in the derivation of its main threshold parameter. For instance it assumes that flows are synchronized following identical congestion avoidance sawtooth (no slow start) and with the same RTT. The recommended queue threshold parameters are ranges and not specific values and this makes it difficult to decide what value to choose. The simulation setup used to validate DCTCP was also too simplistic to show the effects of these assumptions. Moreover, DCTCP trades off convergence time; the time required for a new flow to grab its share of the bandwidth from an existing flow with a large window size. They argue this is OK as DCTCP is designed for short RTT networks. So DCTCP cannot for instance be a good fit for scenarios where a main cloud controls the communication to cloudlets or customer networks at a considerable distance.

1.2.4 Existing DoS and DDoS Mitigation Techniques

The growth in size and complexity of networks also brings other challenges due to another distributed system called Distributed Denial of Service (DDoS). This has prompted extensive research in the area. Research efforts to deal with DoS and DDoS can be classified into different categories such as network filtering and capabilities are discussed in [9]. Network filtering [42] aims to ensure that all traffic for a specific destination (e.g., one under attack) flows through certain filtering boxes. This centralized scheme is difficult to scale in order to deal with an increasing number and sophistication of attacks. On the other hand, Capability approaches [9, 51, 52] drop or send to a lower priority queue every packet that does not carry a certificate that proves that the packet is legitimate. Capabilities are usually created during the connection setup by each router in the path between the source and destination. Each packet carries at least 8-byte capabilities for each router to stamp time-limited cryptographic information. If the destination approves the capabilities-carrying packets, it sends them back to the sender. The sender then attaches the capabilities to each packet it sends so that each router in the path can verify them. However, if the network path changes, the legitimate packets can be treated as attack packets and are either dropped or sent to the lower priority queue. Besides, as explained by Gunes et al. [31], capabilities suffer from a lot of computational, storage and traffic volume overheads. Source and destination authentication may also not help since many colluding sender and receiver nodes, for instance, can cause a serious DoS attack. Other correlation based approaches such as BotHunter [29], BotSniffer [30] and BotMiner [28] for botnet detection suffer from false positives.

1.3 Our Proposed Solutions for the Above Challenges

A lot of research has been done to address the congestion control and routing challenges. All existing works approach each of these problems separately coming up with a congestion control protocol and a routing protocol independent of the congestion control protocol. To deal with the four challenges discussed in the previous sections, we use *a cross-layer congestion control and routing scheme* called the Network Control Protocol (NCP) [23] along with some other efficient algorithms. In this section we discuss how we use NCP with some other elegant algorithms to solve each of the above challenges.

1.3.1 For Congestion Control and Routing in General

Our NCP approach derives a simple and effective congestion control and routing metric. The metric serves as a rate at which sources send data and also as a link metric of a *max/min routing* scheme [17]. Unlike TCP this rate metric can obtain the maximum link utilization and the lowest queue size and hence resulting in the lowest average file completion time (AFCT). It can also be a processor sharing scheme and hence fair to all flows unlike XCP. NCP also uses an exact derivation of the number of active flows and hence doesn't suffer

from such estimation errors of RCP. Unlike RCP, NCP is also not sensitive to the values of many stability and other parameters.

NCP can fall into the clean slate category. However, noting the difficulty to implement pure clean slate protocols in the current networks, we have also designed schemes which can allow NCP to be implemented in the current networks with a few changes at the edge networks [14, 21], and by using the ECN bits of packets [23]. For brevity we will not discuss them in this proposal. However, we will present them in sufficient detail in the PhD thesis.

NCP also has a feature which can achieve Quality of Service (QoS) through proportional allocation as shown in [13]. NCP was called Fast Congestion control Protocol (FCP) [13] in our previous work as it is the fastest such protocol to converge to fairness and to full link utilization. In NCP with QoS feature enabled, each data source can specify its priority level and get a corresponding rate allocation by keeping the salient features of NCP in maintaining (on average) zero queue size (no packet drops) and the minimal possible AFCT. So if one wants higher rate for its special flow, it can set the flows's NCP weight to the ratio of its desired rate to the flow's current rate.

Initial simulation results show how NCP outperforms the other approaches. We have also made initial stability analysis to show that NCP is a stable protocol irrespective of the choice of parameters. In this proposal we will omit the stability analysis using fixed point theorems we used in [13]. We will however discuss the Lyapunov Stability of NCP in this proposal. We also plan to extend this analysis in the final PhD thesis.

1.3.2 For Large Scale Distributed Architectures

We have also extended NCP cross-layer routing and congestion control approach to cloud-computing architectures [18, 19, 20]. A similar cross-layer metric is used to decide where to store data, where to retrieve it from (in case of replicated data), how to route it and at what rate to transfer data from one node to another node. The data transmission can be between nodes in the cloud, or between a node in the cloud and a cloudlet (a mini cloud), between a node in the cloud and a customer device outside the cloud data center.

Our Efficient Distributed File System (EDFS) also has a design feature which solves the single bottleneck name-node scenario of the current state-of-the-art file systems such as the GFS [27] and HDFS [48]. Our design uses a light weight front-end server to forward requests to multiple name nodes which contain detailed information of chunks stored in the block servers (data nodes). Even though our EDFS mechanism can be implemented by making minor modifications in switches and routers and to the TCP/IP stack, we have also designed EDFS in such a way that no changes to the switches, routers or the TCP/IP packet header are needed.

We also have a simple extension of EDFS (GreenEDFS) to make it more energy efficient [15] and needs detailed experimental studies to measure the performance. For brevity the discussion of GreenEDFS is left for the PhD thesis. As mentioned in [18] EDFS can deal with multi-resource bottlenecks. Th storage, processing and link capacity can be the bottleneck resources. The discussion of multi-resource EDFS is also left for the PhD thesis.

For cloudlet architectures which give immediate better data processing, storage and faster connectivity service for mobile wireless devices, we have presented a simulation study [16] and analytical model [36] which determine the number of cloudlets (servers) needed to give full coverage. This analytical model which ensures full coverage to the mobile devices is a multivariate function of the movement area dimensions and other metrics. The performance gains of the congestion control component of EDFS are shown using some initial results which we will discuss later in this proposal.

Our Multiple Resource Bottleneck Rate Allocation Scheme

Our resource allocation scheme with link capacity as a resource constraint can be extended to scenarios where the resource bottleneck is a mix of multiple resources such as link, storage and processing capacities. In this work we also present a generalization of our link rate allocation scheme to apply to other resources.

1.3.3 For DoS and DDoS Mitigation

We have also presented an extension of our cross-layer congestion control and routing approach to solve the fourth challenge related to security discussed above [22]. We believe that if every packet is accounted or paid for, then the DoS and DDoS problem reduces into a congestion control and fairness problem. This problem can then be dealt with by finding better routes or adjusting the sending rates. Hence we design a Packet Accounting System (PAS) to deal with DoS and DDoS attacks. PAS can be implemented by an AS or Internet Service Provider (ISP) to hold the originator of each packet accountable for the congestion it causes.

PAS is a simple and distributed scheme which does not involve senders and receivers unlike the existing well-known capability schemes. PAS router will not need any extra overhead or cryptographic stamp on the packets and hence will avoid the extra computational and traffic volume overhead existing solutions such as the traffic validation architecture (TVA) [52] and Phalanx [9] require. Since each router acts independently of other routers and the end-hosts, a PAS router, unlike TVA and Phalanx, does not require that packets follow the same path. This is especially important since routers do not need capabilities (they need only the IP address, for example) in PAS router to know whether or not a given flow is legitimate. Hence, PAS will not mark legitimate traffic that takes a different route as attack traffic.

As it is expensive to keep and maintain a queue for each flow, TVA uses a hierarchical fair queueing (HFQ) [6] to approximate fair queueing (FQ). A PAS router, however, achieves exact processor sharing or FQ without the complexity of storing, classifying and sequencing packets, which FQ and HFQ suffer from. Preliminary simulation results show how PAS can outperform TVA, a well known DoS mitigation technique which is a Capability approaches as discussed in section 1.2.4. For brevity the discussion of PAS and the relevant preliminary results are reserved for the final PhD thesis.

1.3.4 Research Statement

In light of the challenges and our proposed solutions discussed above the research statement of this work reads as follows:

There exist joint cross-layer routing and congestion control schemes for distributed systems using a metric which offers a fair (equal and proportional) share to flows resulting in minimal average file download (transfer) time. Some of the schemes do not need changes to the existing network infrastructure and others can be implemented with minimal changes to existing networking components. The scheme can result in an accountable network design which limits the effects of DoS and DDoS attacks.

1.4 Proposal Organization

The rest of this proposal is organized in such a way that we first in Chapter 2 present our proposed solution to the first and second challenges discussed in section 1.1 which are applicable to general networks. We then discuss the large scale distributed (cloud computing) communications architecture in Chapter 3 to deal with the third challenge discussed in section 1.1. Then in the last Chapter 4 we present our research plan.

Chapter 2

Schemes for General Networks

In this chapter we present our design of the Network Control Protocol (NCP) which is a cross-layer routing and congestion control scheme and show how it can be used for general networks like the Internet or any Autonomous System (AS) or Internet Service Provider (ISP) level network. We begin with the derivation of the NCP cross-layer metric in section 2.1. In section 2.2 we describe how the NCP mechanism can outperform XCP and RCP which are the other major clean slate congestion control protocols by showing their weaknesses. The NCP algorithm and simulation results to evaluate the performance of the congestion control component of NCP are presented in sections 2.3 and 2.4. Some cross-layer analysis is given in [17] and omitted from this proposal for brevity and will be included in the final thesis. We present stability analysis of NCP in section 2.5. Finally a brief chapter summary is given in section 2.6.

2.1 Cross-layer Metric and its Derivations

In this section we will first derive the fair share metric and then we will generalize it to proportional share rate metric which can prioritize flows to achieve the desired QoS level given the capacity constraint.

2.1.1 Fair Share Metric

To define and derive the NCP rate metric for general networks, we first present the following notations in table 2.1.1.

Variables	Description
C	Link capacity
d	Control interval
$q(t-d)$	Link queue size from the previous interval (round)
$q(t)$	Link queue size from the current interval (round)
$R(t-d)$	Link rate allocation of the previous interval (round)
$R(t)$	Down-link rate allocation of the current interval (round)
N	Number of flows in the link during the current round
L	Total number of packets which arrive to the router during a the control interval, d
w_j	The current *cwnd* (congestion window) carried by packet j
ϵ_j	The size of packet j
α, β	Stability parameters

Table 2.1: NCP Variables for General Networks

Given the above notations in table 2.1.1, the per flow fair NCP rate allocation at a bottleneck router is derived as follows. Let w_j be the current *cwnd* (congestion window) of a flow attached to the jth packet of the L packets which arrive at a router during the control interval d and which is used to calculate the throughput $R(t)$ and the *cwnd* w'_j for the next round. Let us define the ***per packet throughput*** to be the number of bytes 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 bottleneck link capacity minus the bandwidth needed to drain the queue within a round trip time (RTT) or within a control interval. That is

$$\sum_{j=1}^{L} \frac{R(t)}{w_j} = \alpha C - \beta \frac{q(t)}{d}. \tag{2.1}$$

This implies that

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d}}{\sum_{j=1}^{L}(1/w_j)}. \tag{2.2}$$

By using the estimation $w_j = dR(t-d)$ in Equation 2.2 the NCP rate can be given by

$$R(t) = \frac{(\alpha C - \beta \frac{q(t)}{d})R(t-d)}{\Lambda} \tag{2.3}$$

where $\Lambda = L/d$ is total packet arrival rate to the router. This can serve as a simplified version of NCP and needs less work at the routers.

The NCP rate can also be derived using the fact 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 is the sum of the *per packet cwnd* which is the number of bytes sent at an arrival of each of the w_j ACK packets. Hence if $R_j = w_j/RTT_j$ denotes the rate attached to the jth of the L packets which arrive to the router,

$$\sum_{j=1}^{L} \frac{\epsilon_j R(t) RTT_j}{w_j} = \sum_{j=1}^{L} \frac{\epsilon_j R(t)}{R_j} = \alpha C d - \beta q(t). \tag{2.4}$$

This implies that

$$R(t) = \frac{\alpha C d - \beta q(t)}{\sum_{j=1}^{L}(\epsilon_j/R_j)} = \frac{\alpha C - \beta \frac{q(t)}{d}}{\frac{1}{d}\sum_{j=1}^{L} \frac{\epsilon_j}{R_j}}. \tag{2.5}$$

After the initial allocation of the rate given by 2.5 at time $t-d$, flows are sending at the rate of $R(t-d)$ given by the same equation 2.5. Hence we can estimate R_j with $R(t-d)$ and equation 2.5 becomes the same as equation 2.3.

Another derivation of the NCP fair share rate is also given in our previous work [13].

Processor Sharing (PS) Using *cwnd*

In this section we will show how the NCP formulas given by equations 2.2 and 2.5 achieve processor sharing (PS).

If we denote the total number of concurrent flows at a router with N, re-arranging the headers of the packets which arrive to the router during a control interval, Equation 2.2 can be written as

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d}}{\underbrace{\frac{1}{w_1} + \frac{1}{w_1} + \cdots + \frac{1}{w_1}}_{n_1} + \underbrace{\frac{1}{w_2} + \frac{1}{w_2} + \cdots + \frac{1}{w_2}}_{n_2} + \cdots + \underbrace{\frac{1}{w_N} + \frac{1}{w_N} + \cdots + \frac{1}{w_N}}_{n_N}} \tag{2.6}$$

which is the same as

$$R(t) = \frac{\alpha C - \beta \frac{q(t)}{d}}{N} \tag{2.7}$$

if $n_i = w_i$ 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 for this case where all flows can send a *cwnd* of packets during the router control interval d.

Efficient Sharing (ES) Using Fractional Flows

Equation 2.6 not only shows how NCP achieves PS but also handles a scenario where PS cannot handle. This scenario occurs when a *fractional number of flows* arrive to the bottleneck link during d. The usual PS counts a flow as one flow even if only half (or some other fraction) of its *cwnd* of packets arrive to the router during d. This can arise when a flow i with a *cwnd* of w_i carried in the packets cannot send w_i packets within the router control interval d. For instance if flow i is bottlenecked in some other links that its packets are delayed, the number n_i of flow i packets which arrive to the bottleneck link router during d is less than w_i. The RTT_i of flow i may also be larger than d causing $w_i > n_i$.

If exact PS is used for a scenario where $n_i < w_i$ then a flow is counted as one flow even though it can only send a very small fraction of what is allocated to it because of its other constraints. On the other hand NCP can count a flow as a *fractional* (partial) flow if it is not a full flow to the router. So the number N_{NCP} of flows during control interval using NCP is given by

$$N_{NCP} = \sum_{i}^{N} \frac{n_i}{w_i} \tag{2.8}$$

where $N = N_{PS}$ with N_{PS} being the number flows according to PS (Processor Sharing) which is a simple count of distinct flows which arrive to the router during d regardless of the number of packets of each flow which arrives to the router during the control interval.

If $N_{PS} > N_{NCP}$ and if N_{PS} is used in equation 2.6 for rate allocation then sources get less than what they should get and hence can result in resource underutilization. The case $N_{PS} < N_{NCP}$ can also result in resource over-utilization if N_{PS} is used. From this argument we can say that PS can waste resources or result in resource overflow. Hence PS is not efficient. So the NCP sharing which we call *Efficient Sharing (ES)* of resources using the concept of *fractional* flows is better than PS.

The idea of ES is different from generalized processor sharing (GPS) approach where flows get a certain weighted share of the resource to benefit some flows 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. The proportional NCP rate metric derived in the next section 2.1.2 is one way of achieving GES. The GES scheme obtains the fair share rate using the concept of *fractional flows* and shares that resource according to the weights.

We next show how the second rate formulation given by equation 2.5 also achieves ES and hence PS.

ES Using Flow Rate

The inter-packet time σ_j is defined as the time between two consecutive packets for flow j, which is given by

$$\sigma_j = \frac{1}{R_j}, . \tag{2.9}$$

Now suppose a router has seen L packets within the control time interval d. If n_i of these packets carrying σ_i from source i are received by the router during the control interval d, then taking the denominator of equation 2.5 we have

$$\frac{1}{d}\sum_{j}^{L}\frac{\epsilon_j}{R_j} = \frac{1}{d}\sum_{i=1}^{N} n_i \frac{\epsilon_i}{R_i} = \sum_{i=1}^{N}\frac{n_i}{d}\frac{\epsilon_i RTT_i}{\epsilon_i w_i} \tag{2.10}$$

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 packets source i sends during its round trip time (RTT_i). The variable n_i is the total number of flow i packets which arrive at the router during the control interval d.

In the case where all packets 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 packets can be spaced at an equal interval of σ_i) we have that

$$\frac{n_i}{d} = \frac{w_i}{RTT_i}. \tag{2.11}$$

This implies that

$$\frac{1}{d}\sum_{j}^{L}\frac{\epsilon_j}{R_j} = N. \tag{2.12}$$

In the case where $\frac{n_i}{d} \neq \frac{w_i}{RTT_i}$, PS fails to give the exact *fractional* number of flows. As a result the PS rate allocation can result in resource underutilization or buffer overflow. However ES can handle such a scenario where we do not necessarily get an integral number of flows in a given control interval.

Another important result from equation 2.11 is that unlike RCP [10] and XCP [37] the estimation of the control interval in NCP *doesn't depend on the RTT* of the flows. It can be set to any reasonable value. This can enable NCP to be implemented without the need of an additional RTT field in its header.

NCP achieving ES in a Multi-bottleneck Network

In a network where different flows are bottlenecked at different links, some flows may not be able to utilize their fair share allocation at a link which is a bottleneck for other flows. If the bottleneck link allocation of flow i is R_i and if its current rate share at its non bottleneck link is $R(t-d)$, then flow i can waste its non-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 2.5 with

$$max(R_j, R(t-d)) \tag{2.13}$$

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

Here is the explanation of why this approach can achieve ES. If $R_j < R(t-d)$, then a flow which owns packet j should be treated as a partial (fractional) flow by the router which allocated $R(t-d)$ to the flows (including the flow of packet j) in order for 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 packet j as more than one flow as it causes 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 to know about its latest rate allocation.

If we approximate R_j used in equation 2.5 with $R(t-d)$ even if $R_j > R(t-d)$, we get

$$N_a = \frac{1}{d}\sum_{j}^{L}\frac{\epsilon_j}{R_j} \approx \frac{1}{d}\sum_{j}^{L}\frac{\epsilon_j}{R(t-d)} \approx \frac{1}{R(t-d)}\frac{\sum_j^L \epsilon_j}{d} = \frac{y(t)}{R(t-d)} \tag{2.14}$$

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

When NCP uses equation 2.14, it can underestimate the actual number N_a of flows when $\frac{\epsilon_j}{R_j} < \frac{\epsilon_j}{R(t-d)}$. This under-estimation of N_a can result in a higher rate allocation to all flows which in turn can result in queue spikes and even buffer overflows. But this queue spike doesn't usually last long as the exact estimation of N_a can be obtained in the next round. Besides NCP drains the queue $q(t)$ before it makes a new allocation.

In a simplified and efficient version of NCP, we also use equation 2.14 in the denominator of equation 2.5 as an estimation of the actual number of flows as also presented in [24]. The derivation in equation 2.14 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 2.14 is a very good estimator of ES. This implementation allows NCP packet header to be even smaller (about 8 bytes) as shown in section 2.3.1.

2.1.2 Proportional Share Metric

As also derived in [13], the NCP rate which is given by equations 2.2 and 2.5 can be extended to be a proportional share metric. Such a proportional metric allows different flows to get different shares based on their weights without causing router buffer overflow or link under-utilization. Different policies can be set

for different classes of flows. For instance, the priority p_j of a flow attached to packet j can be between p_{min} and p_{max}. In this case, a flow to which packet j belongs gets the share

$$R_j(t) = p_j R(t)$$

where $R(t)$ is derived as follows.

We make this derivation with the same notations we used above where $R_j = w_j/RTT_j$ is the rate attached to the jth of the L packets which arrive to the router. Using the fact that the total number of bytes sent to a router (link) during a control interval d shouldn't exceed the bandwidth-delay product minus the queue size at the router,

$$\sum_{j=1}^{L} \frac{\epsilon_j p_j R(t)}{R_j} = \alpha C d - \beta q(t). \tag{2.15}$$

This implies that

$$R(t) = \frac{\alpha C d - \beta q(t)}{\sum_{j=1}^{L} (p_j \epsilon_j / R_j)}. \tag{2.16}$$

Another derivation of the NCP fair share rate is also given in our previous work [13].

Different levels of priority can be used by adding a few more bits in the NCP header or using the current IP header fields (ECN bits). The source can also send $p_j\epsilon_j/R_j$ in the NCP header. Each source i can then set its congestion window as $w_i = p_i R_i RTT_i$ packets where R_i is obtained from the ACK packets. Since routers can get the packet size, the NCP header can also carry only p_j/R_j in the case of proportional share and $1/R_j$ in NCP implementation with out weights.

2.2 NCP Versus Other Major Clean Slate Protocols

In this section we discuss how NCP differs from RCP and XCP which are the two other major clean slate congestion control protocols.

2.2.1 On Performance of RCP

The rate update equation of the newly proposed rate control protocol (RCP) [10] for the Internet is given by

$$R(t) = R(t-d) + \frac{(\alpha(C - y(t)) - \beta \frac{q(t)}{d})}{N(t)} \tag{2.17}$$

where d is a moving average of the RTTs measured across all packets, $R(t-d)$ 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 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) [12], the number of ongoing flows, $N(t)$ is estimated as

$$N(t) = \frac{C}{R(t-d)}. \tag{2.18}$$

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

So RCP either over-estimates or under-estimates the allocated rate $R(t)$. When the initial value of $R(t-d)$ 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 $R(t-d) = C/200$ 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 $R(t-d)$ 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 $R(t-d) = C/4$. 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 $R(t-d) = C/4$. 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.

In section A.1, we explain why RCP seems to closely emulate processor sharing in the published literature while that is not the case. In section A.2 we summarize the scenarios where RCP works and doesn't work well by deriving it using NCP. NCP on the other hand gives exact derivation of the actual number of flows $N(t)$.

2.2.2 On 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 [11]. 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 for most of the flows. But NCP can reduce this FCT of majority of the flows 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 discuss more about how the rate allocation scheme of XCP differs from that of NCP in section A.2 by deriving XCP using NCP.

2.3 NCP Algorithm

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

- A source sends a packet j with its desired rate R_j.
- Each router in the network calculates $R(t)$ which is given by equation 2.5 or equation 2.3 every control interval d, $0 < d \leq RTT_{max}$. Here RTT_{max} is the maximum RTT of the flows which can be known before hand.
- Each router in the path of packet j 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 d.

At some user defined regular intervals the routers can also use a max/min routing algorithm like the one in [17] to find the best path for the packets in the network.

2.3.1 NCP Packet Header Format

The NCP protocol can have two packet header implementation schemes. The first one which is shown in figure 2.1 has a 12 byte header.

The first field is the *Inter-Packet Interval* length $\ell_j = 1/R_j$, where R_j is the current sending rate attached to packet j of a flow. The routers in the path of packet j use this field to obtain the NCP rate given by

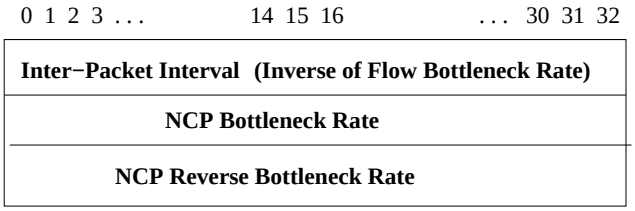


Figure 2.1: NCP header with 12 bytes

equation 2.5. The second field is the *NCP Bottleneck Rate* $R(t)$ (equation 2.5) which is the rate initialized to be the desired rate by source. The bottleneck router in the path of the packet j can then overwrite the value. This rate is the minimum of all the rates in the path of packet 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 (general network) used in this proposal use this implementation scheme of the NCP header.

The second implementation scheme of the NCP header is shown in figure 2.2 is without the ℓ_j field. This implementation can reduce the NCP packet header to 8 bytes.

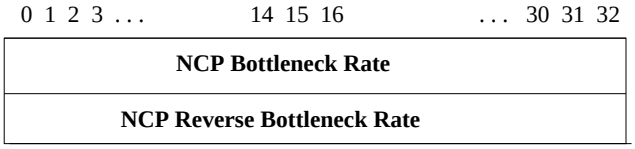


Figure 2.2: NCP header with 8 bytes

In this implementation scheme each source sets the value of the *NCP bottleneck rate* (R_j) to its desired rate. Each router in the path of packet j calculates the rate using equation 2.14. If this rate is smaller than the R_j in the packet header, then the router replaces the R_j in the packet header with what it obtain using equation2.14. 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.

2.4 Congestion Control Analysis: RCP Vs NCP

In this section we evaluate the performance of NCP comparing it with RCP using NS2 [43] which is the state of the art network simulator. We first briefly overview the implementation of NCP in NS2 in Section 2.4.1. We then show simulation results and analysis in Section 2.4.2.

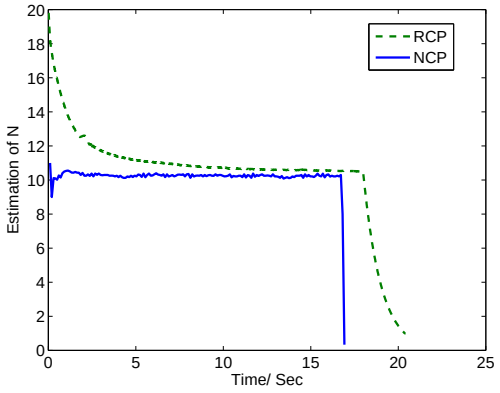
2.4.1 NCP Implementation in NS2

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. We coded NCP source in two ways. One way is by sending a window of data packets back-to-back upon reception of an ACK which carries the window size information. The second is by spacing packets by an inter-packet time which is the inverse of the rate allocation of the flow as shown in Equation 2.9.

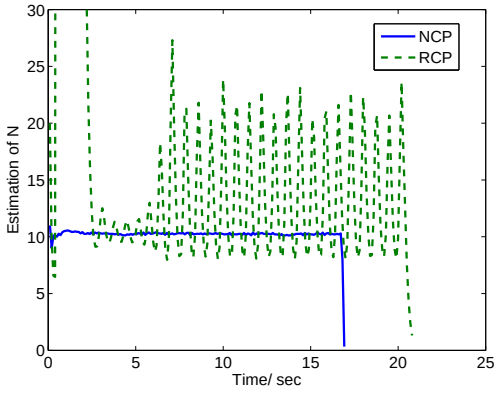
A common implementation of TCP also sends packets back-to-back upon reception of an ACK. Sending a window of packets back-to-back may be an energy saving strategy for mobile wireless devices as they do not have to stay in an active mode sending packets during the entire round trip time. This scheme however resulted in non-uniform arrival of packets to the NCP queue. This in turn resulted in unstable estimation of the number N of flows used in the calculation of the NCP rate shown in Equation 2.12. So in our experiments, we used the approach where 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.

Table 2.2: Baseline parameters for experiments on estimation of N

Parameter	Default value
Link capacity	20 Mbps
Link propagation delay	0.05 second
Number of flows	10
File size	4 MB



(a) Estimation of N versus time with $\alpha = 0.1$ for RCP



(b) Estimation of N versus time with $\alpha = 1$ for RCP

2.4.2 Simulation Results

We have conducted extensive simulations on NS2 to compare the performance of NCP and RCP. Similar to previous work on RCP, we use a simple topology which contains a sources and a destinations connected by one single link.

In the first set of experiments, show how the two protocols make estimation on the number of flows. For conciseness, we generate a fixed number of big size flows which all start at the same time. The baseline parameters are summarized in Table 2.2.

Figure 2.3(a) and Figure 2.3(b) plot the estimation of number of flows versus time for NCP and RCP. The first and later plots use different settings of α for RCP. The estimation of N from NCP virtually matches the real value. In contrast, depending on the choice of parameters, the estimation of N from RCP either needs much longer time to converge or even never converges. Two important messages conveyed here are:

- NCP gives much more accurate and reliable estimation of the number of flows than RCP;
- the performance of RCP is sensitive to the setting of parameters.

To compare the performance of NCP against RCP we have also considered a fixed number of flows with a fixed file size.

In the next set of experiments, we compare RCP and NCP with respect to the flow completion time. Flows are generated at the source following Poisson distribution. The file size of the flows is generated using Pareto distribution. The baseline parameters used in this set of experiments are summarized in Table 2.3.

Figure 2.4(a) plots the cumulative distribution function (CDF) of the Pareto file size distribution with mean 25 and shape parameter of 1.2 used in our simulation. The majority of the flows has a small file size.

Figure 2.4(b) plots the CDF of the FCT distribution. This figure shows that majority of the flows complete faster in NCP than in RCP.

The figure 2.4(b) plot was obtained from a simulation setup where on average the link is only 90% loaded. In this case, Poisson flow arrival of 10800 flows/sec where the file size is Pareto distributed with an average of 25 packets is used. Most of the experiments used to validate RCP in the literature were similarly obtained using a non-congestion scenario. As discussed in section A.1, this 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

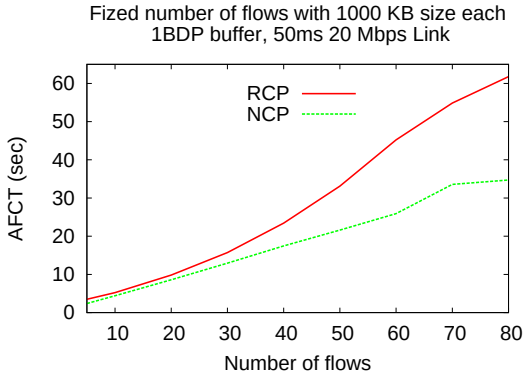
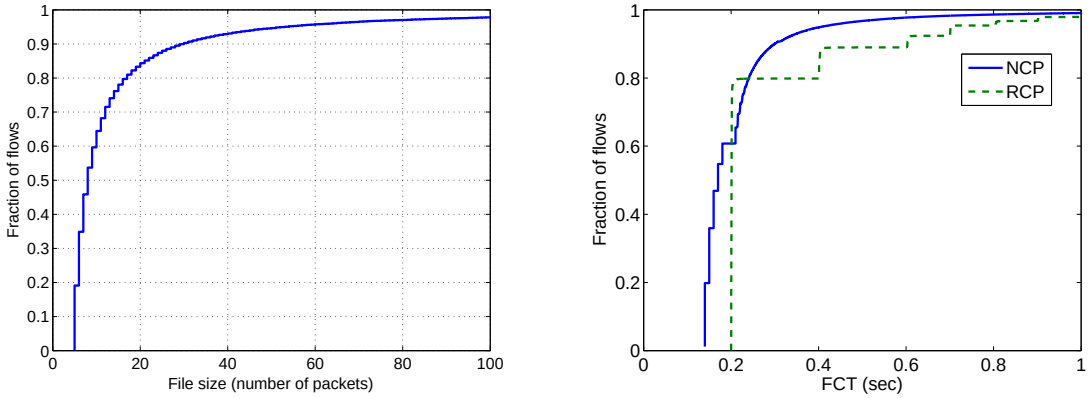


Figure 2.3: AFCT versus number of flows

Table 2.3: Baseline parameters for experiments on FCT

Parameter	Default value
Link capacity	2.4 and 1.2 Gbps
Link propagation delay	0.05 seconds
Mean flow arrival rate (Poisson)	10800 flows/sec, 3000 flows/sec
Mean file size (Pareto)	25 KB, 500 KB
Shape parameter (Pareto)	1.2



(a) CDF of file size

(b) CDF of FCT

Figure 2.4: Comparison of NCP and RCP with respect to FCT.

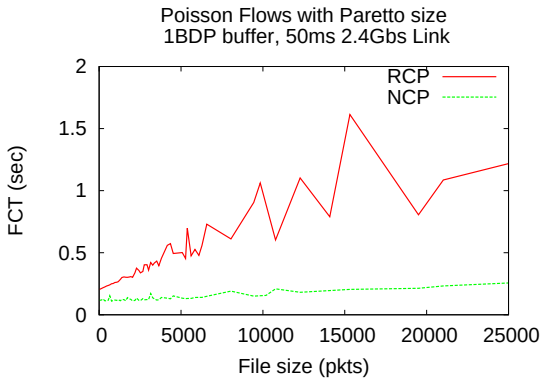


Figure 2.5: AFCT of *Naive NCP* vs RCP

Table 2.4: NCP versus RCP under high load scenario: Poisson(3000), Pareto(1.2,500)

Protocol	Number of finished flows (in 68.5 sec)
RCP	16167
NCP	5340

of the flow for the cases where the link on average is not fully utilized (similar to most RCP experiments in the literature), the network doesn't get congested on average as shown in figure 2.5. In this scenario a congestion control protocol is not even needed as all flows can send all the packets they have in one one round and retransmit some of their lost or delayed packets to get very small AFCT. As can be seen from the plot NCP outperforms RCP.

However under a real congestion scenario the performance of RCP is a lot worse when compared with NCP as shown in the next experimental results. In this experiments a 3000 flows/sec average Poisson flow arrival where the file sizes are Pareto distributed with a mean of 500 packets and Pareto shape of 1.2 is used. As shown in table 2.4, within a simulation time of 68.5 seconds only 5340 RCP flows finished due to the increasingly high file completion time (FCT) as shown in figures 2.6(a) and 2.6(b). On the other hand as can be seen from table 2.4 16167 NCP flows finished during the same time.

We also compared the FCT of the 5340 RCP flows (all RCP flows) which finished against the first 5340 NCP flows which finished. As shown in figure 2.7 the FCT of NCP flows is much smaller than that of RCP. This small FCT helped more NCP flows finish in a shorter time as shown in table 2.4.

2.5 Stability Analysis

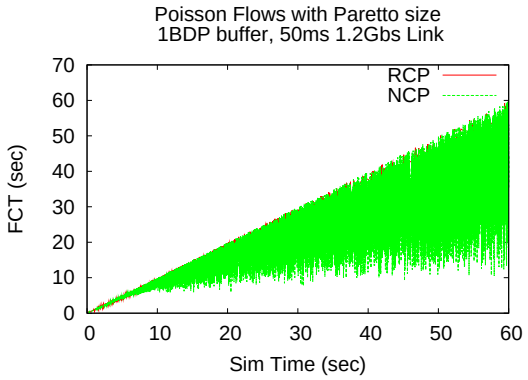
In this section we present stability analysis using some control theory. In the PhD thesis we will also discuss stability analysis using the fixed point theory as shown in [13].

2.5.1 Lyapunov Stability

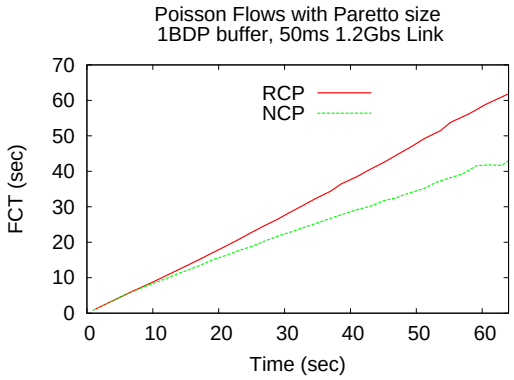
The rate allocation by an NCP queue at a bottleneck router is done every control interval d. This allocation is received by each source sharing the bottleneck link after a round trip time (of each of the sources). This new rate allocation changes the congestion window w_j of each source j. So the aggregate 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

$$\sum_j \frac{dw_j}{dt} = C - \Lambda(t-d) - \frac{q(t-d)}{d} \tag{2.19}$$

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.



(a) FCT of flows versus simulation time



(b) FCT of flows versus simulation time (with 2 sec aggregation)

Figure 2.6: FCT versus simulation

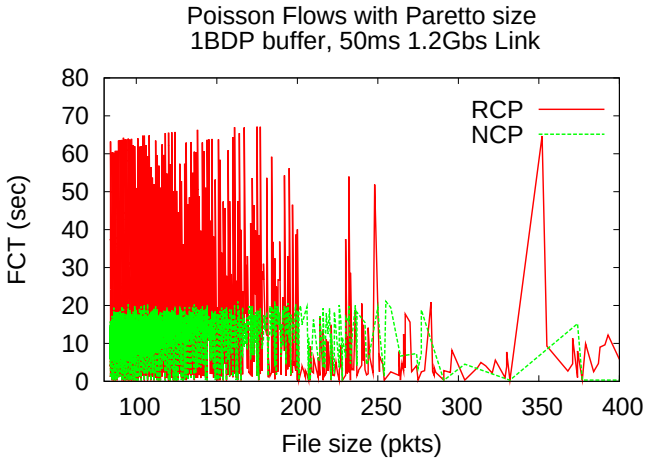


Figure 2.7: FCT of finished RCP flows vs NCP flows

Adding the control parameters α and β for stability, Equation 2.19 becomes

$$\sum_j \frac{dw_j}{dt} = \alpha(C - \Lambda(t-d)) - \beta\frac{q(t-d)}{d}. \tag{2.20}$$

As shown in [5] and [41], the NCP feedback mechanism is given by the delay differential equations

$$\begin{aligned} \Lambda'(t) &= \frac{\alpha}{d}(C - \Lambda(t-d)) - \frac{\beta}{d^2}q(t-d) \\ q'(t) &= \begin{cases} \Lambda(t) - C, & q(t) > 0 \\ max\{\Lambda(t) - C, 0\}, & q(t) = 0. \end{cases} \end{aligned} \tag{2.21}$$

As the NCP feedback mechanism can be written in Equation 2.21, appropriate Lyapunov functions can be used to find stable values of the control parameters α and β. For instance as the NCP feedback mechanism can be written in the form given by Equation 2.21, the work [41] shows that $\beta/d^2 = \alpha/d$ gives stability. This for instance implies that if $\alpha = 1.0$, $\beta = d$. Previous work [5] also shows a wide range of stable values for protocols whose feedback mechanism can be written in the form of Equation 2.21. 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 [41] to find even wider stable regions for α and β.

2.6 Summary

In this chapter we presented the design of a cross-layer congestion control and routing network control protocol (NCP) when used in general networks like the Internet. NCP uses a *fair rate metric* to determine *the rate at which flows send data*. NCP also uses the rate metric as a link weight to find *the high throughput path* for the flows using a max/min routing scheme. NCP can also provide a *proportional allocation* where some flows can get higher bandwidth than others.

We have discussed how NCP can outperform other major clean slate congestion control protocols like the RCP and XCP. We have also made simulation of analysis showing that the congestion control component of NCP can outperform RCP. We have also made initial stability analysis of NCP describing that NCP is stable for a wide range of parameter settings.

Chapter 3

Schemes for Large Scale Distributed File System (Cloud Computing)

In this chapter we present a large scale distributed systems architecture. The scheme which we call Efficient Distributed File System (EDFS) relies on the NCP discussed in the previous Chapter 2 to decide the rate at which nodes exchange data. A scheme similar to the cross-layer feature of NCP is also used by EDFS to decide where in the distributed system (cloud) to store data. EDFS using NCP selects servers which minimize file transfer time.

The design of EDFS has two main features. The first feature enables EDFS to use multiple name node servers (NNS) using a light weight front-end server (FES) which forwards requests to the name nodes (NNS). This approach solves the weakness of current state-of-the-art cloud-computing architectures (file systems). In such systems only a single NNS, which can potentially be a bottleneck resource and single point of failure, is used.

The second main feature of EDFS is its ability to avoid congestion and select the less loaded servers using the NCP cross-layer concept. EDFS also uses resource monitors and resource allocators to do fine grained resource allocation and load balancing. The roles of these EDFS components can be extended to constantly monitor the performance of the cloud against malicious attacks or failures. All the aggregated and monitored traffic can be offloaded to an external server for off-line diagnosis, analysis and data mining of the distributed system.

The rest of this chapter is organized in such a way that we first discuss the EDFS architecture in section 3.1. In section 3.2 we explain the EDFS rate metric. In section 3.2 some discussion of the EDFS algorithm with only link capacity as a bottleneck resource is given. A brief description of a multi-resource QoS aware rate metric extension of NCP is also given in section 3.4. We give a brief discussion of an analytical model to find the number of cloudlets needed to give wireless network coverage to an area in section 3.5. We present simulation results to evaluate the performance of congestion control component of EDFS in section 3.6. Finally we summarize the chapter in section 3.7.

3.1 EDFS Architecture

The architecture of EDFS [20] is presented in Figure 3.1. As shown in the figure, the EDFS architecture we present in this proposal assumes a tree structure of the data center networks for cloud computing as is the case with most data center networks today. However, our EDFS scheme works with any cloud and data center topology. Like existing popular large scale distributed file systems [27, 48], EDFS consists of a network of block servers (BS). Unlike GFS and HDFS, EDFS uses a light weight front end server (FES) and more than one name node server (NNS). This enables EDFS to solve the potential problems of GFS and EDFS in being bottlenecked at the single NNS. EDFS also achieves its efficient resource allocation and load balancing schemes and energy efficiency using rate monitors and rate allocators. We next discuss the nodes and the resource monitors and allocators of EDFS.

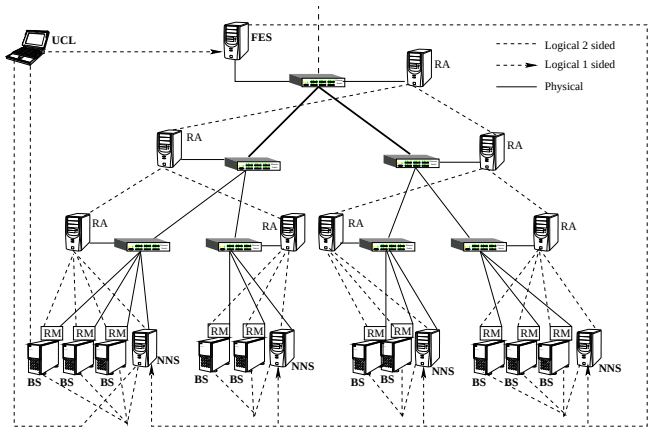


Figure 3.1: EDFS Architecture

3.1.1 Nodes

The nodes in EDFS consist of the front end server (FES) which receives external requests to and from the local cloud and forwards them to the respective name node server (NNS). Each NNS keeps metadata information, for example, which block of data is stored in which block server (BS). Each BS stores data blocks assigned to it by the NNS. To help balance load among all NNS, the FES may also be assisted by the NNS to forward requests to other NNS. The UCL node is a user client which requests cloud services.

3.1.2 Resource Monitors and Allocators

The resource monitor (RM) of EDFS is a software component responsible for monitoring and sending resource load information from the BS to the resource allocators (RA). The RA on the other hand gathers resource load information from each BS via the RMs and other information from the switch and calculates a EDFS rate allocation metric at each level of the tree.

3.2 Calculation of the EDFS Rate Metric

To define the rate metric, we first present the following notations.

For each EDFS parameter $X \in \{R, C, Q, \hat{N}, N, n^j, R^j\}$,

$$X_{d,u} = \begin{cases} X_d & \text{if } X \text{ is a downlink parameter,} \\ X_u & \text{if } X \text{ is a uplink parameter.} \end{cases} \tag{3.1}$$

We next give short descriptions of the EDFS parameters.

Variables	Description
$C_{d,u}$	Link capacity
d	Control interval
$Q_{d,u}(t-d)$	Link queue size from the previous interval (round)
$Q_{d,u}(t)$	Link queue size from the current interval (round)
$R_{d,u}(t-d)$	Link rate allocation of the previous interval (round)
$R_{d,u}(t)$	link rate allocation of the current interval (round)
$N_{d,u}(t-d)$	Number of flows in the link during the previous round
$N_{d,u}(t)$	Number of flows in the link during the current round
$\hat{N}_{d,u}(t-d)$	Effective number of flows in the link during the previous round
$\hat{N}_{d,u}(t)$	Effective number of flows in the link for the current round
$n^j_{d,u}(t-d)$	Link flow indicator of the previous round
$R^j_{d,u}(t)$	Link rate allocation of flow j for the current round
$\Lambda_{d,u}(t)$	Total current arrival rate to the link
$L_{d,u}(t)$	Total number of packets in the current interval

Table 3.1: EDFS Parameters

Given the above EDFS parameters, each RA and RM calculate the rates $R_d(t), R_u(t)$ of the down (d) and up (u) links associated with their local switches as follows:

The down-link (d) and up-link (u) rates

$$R_{d,u}(t) = \frac{C_{d,u} - \frac{Q_{d,u}(t-d)}{d}}{\hat{N}_{d,u}(t-d)} \tag{3.2}$$

where

$$\hat{N}_{d,u}(t-d) = \sum_{j}^{N_{d,u}(t-d)} n^j_{d,u}(t-d) \tag{3.3}$$

and

$$n^j_{d,u}(t-d) = \begin{cases} \frac{R^j_{d,u}(t)}{R_{d,u}(t-d)} & \text{if } R^j_{d,u}(t) < R_{d,u}(t-d), \\ 1 & \text{otherwise.} \end{cases} \tag{3.4}$$

As shown in Figure 3.1, each RA and RM get the values of $Q_d(t-d)$ and $Q_u(t-d)$ from the local switch (router) to which they are connected. This doesn't need any change to the switches as all switches maintain the queue length in each of their interfaces. Each RM computes the effective number of up-link and down-link flows using equation 3.3. Each RM reports the values of $\hat{N}_d(t-d)$ and $\hat{N}_u(t-d)$ to its parent RA. Each RA adds these values from each of its children to find its $\hat{N}_d(t-d)$ and $\hat{N}_u(t-d)$ values. Each RA also sends its accumulated effective number of flows for both the down-link and up-link to its parent RA. This continues until the highest level RA. After the first time a RM sends its $\hat{N}_d(t-d)$ and $\hat{N}_u(t-d)$ values, it can send the difference Δ_d and Δ_u values to its parents for all other rounds. This is to minimize the overhead by sending the difference which is a smaller number than the actual number of flows. Each RA also does the same by sending the difference instead of the actual effective number of flows to its parent RA.

Each RM and RA perform the computation of equation 3.2 periodically every control interval d. This control interval for the RM can be estimated as the average of the round trip times (RTT) of the flows of its BS or it can be a user defined parameter. Each RA at level h can compute its $R_{d,u}(t)$ after it gathers the $\hat{N}_d(t-d)$ and $\hat{N}_u(t-d)$ information from all its children or after a certain time out value T_o expires.

The simplified version of NCP given by equation 2.3 which we have shown to closely emulate ES (efficient sharing) can also be used as an EDFS rate metric. So in this case the simplified EDFS rate metric is given by

$$R_{d,u}(t) = \frac{(\alpha C_{d,u} - \beta \frac{Q_{d,u}}{d}) R_{d,u}(t-d)}{\Lambda_{d,u}(t)} \tag{3.5}$$

where $\Lambda_{d,u}(t) = L_{d,u}/d$ is total packet arrival rate to the router. In this simplified version of EDFS each RA and RM can also get the values of $L_{d,u}$ from the corresponding switch or router. Hence for this simplified version of EDFS the RMs and RAs do not need to report their estimate of the actual number $\hat{N}_d(t-d)$ and number $\hat{N}_u(t-d)$ of flows to their parent nodes (RA).

3.3 EDFS Algorithm (Only Link Capacity as a Bottleneck Resource)

In this section we will describe how the EDFS algorithm works by using only the link capacity as a bottleneck resource. The analysis when other resources are bottlenecks is left for future work.

3.3.1 Global and h-Level Rate Allocation

Each NNS needs to decide which BS at level h to chose for each block of data and at what rate to send data from one BS to another BS or to/from an external agent. The NNS then asks the RA at level h, $1 \leq h \leq h_{max}$ of the tree as shown in Figure 3.1. Hence each RA needs to maintain the best down-link $\hat{R}_h$ and up-link $\check{R}_h$ rate values and the address of the BS or BSes with these rate values. For global allocation, the highest level

values are needed. Here h_{max} is the maximum level value in the tree like cloud topology starting from the BS nodes.

Each NNS among other things also needs to decide at what rate to copy data from one BS in one level of the cloud tree to another BS in another part of the cloud by asking each RM. Hence each RM also needs to keep the up-link and down-link bottleneck rate value upto each level of the tree. To achieve this each RA forwards its rate values obtained using equation 3.2 to its child. Besides, each RA needs to forward to its children the minimum of its rate and the rates forwarded to it from its higher level parents. Finally, these rates of each level of the cloud tree are received by each RM.

The above best h-level rate values stored at each RA and RM are obtained using a max-min scheme as follows.

Obtaining the Rate values using Max/Min Algorithm

Here is how the rate metric at different levels of the network tree are obtained.

- Each RM at level $h = 0$ sets its up-link $\check{R}_h$ rate value to its $R_u(t)$ which is obtained using Equation 3.2.
- Each RM sends its $\check{R}_0$ to its parent RA which is associated with the switch the RM and RA are directly connected to.
- Each RA at level h sets its $\hat{R}_h$ to the minimum of its $R_d(t)$ obtained using Equation 3.2 and the highest $\hat{R}_{h-1}$ obtained from its children. There is no down-link rate $\hat{R}_h$ obtained from the RMs as there is no link going down from them (the BSes). So each RA which is a parent of an RM just obtains its $\hat{R}_h$ using Equation 3.2. Similarly, it sets its up-link $\check{R}_h$ to the minimum of its $R_u(t)$ obtained using Equation 3.2 and the highest $\check{R}_{h-1}$ obtained from its children. The RA then stores its $\hat{R}_h$ and $\check{R}_h$ values and sends them to its parent RA along with the ID of the corresponding BS. The parent also does the same.
- By the time this process reaches the RA at level h_{max} which is the highest level RA at the entry point to the cloud, each RA at level h has the best h-level $\hat{R}_h$ and $\check{R}_h$ and the ID of the corresponding best BS. These values are useful for the NNS to decide *where to store (write) data*.
- The highest level RA (at level $h = h_{max}$) sends its $R_d(t)$ and $R_u(t)$ values along with its level number down to its children RAs. Each RA at level h also forwards the minimum of its rate and each of its higher level rates along with the level numbers to its children. Finally each lowest level RA forwards these values to its children RM.
- At this point each RM knows the best h-level up-link and down-link rate values along with the level numbers. These values are helpful for the NNS in deciding where *to read data from* and how to update the *rates of on-going flows* to and from the main cloud (data center) using the information in the RM.

Allocation for Interactive Applications

The above allocation scheme is good for cases where the allocation scheme finds the BS with the best upload rate values or best download values in each branch of the tree-like cloud architecture. However some applications may be interactive with the read weight of w, $0 \leq w \leq 1$ and write weight of $1 - w$. In this case each RA keeps the rate $\tilde{R}_h(w) = w\check{R}_h + (1 - w)\hat{R}_h$. Here it can be seen that for the pure download case (all writing) $\tilde{R}_h(0) = \hat{R}_h$ and for the pure upload (reading case), $\tilde{R}_h(1) = \check{R}_h$. To minimize the storage and computational overhead, only a few common weights can be considered. For instance we can consider the weights $w \in \{0, 0.25, 0.5, 0.75, 1\}$. The EDFS design is flexible in trading off on how many weights can be used. For instance only the weights $w \in \{0, 0.5, 1\}$ can be used.

In this scheme with interactive weights, each RA at level h sets its $\tilde{R}_h(w)$ to the minimum of its $\tilde{R}_h(w)$ and the the highest $\tilde{R}_{h-1}(w)$ obtained from its children. It stores this value and the ID of the corresponding BS. It then sends its $\tilde{R}_h(w)$ to its parent and the parent does the same.

Each application which uses the cloud resources can specify the weight value w. The BS can also estimate this value from the frequency of reads and writes of the specific application and send it to its RM.

Where to store data and at what rate?

Once the RAs of EDFS keep the ID of the BS with the best upload and download rates at different levels of the EDFS topology, the NNS can decide to store data at the best BS at the rate it obtains from a specific RA based on its policies. The requests to store, read or update data can be *external* or *internal*. We next discuss these two different types of requests.

Serving External Requests

An external request for cloud resources such as writing to the cloud, reading from the cloud or doing computation in the cloud using a user client (UCL) may use a dedicated tunnel included in the cloud services. For instance the UCL can be a cloudlet or private network managed by the cloud admins. The request may as well have to cross a network which the cloud administrator has no control over.

In the first case where the cloud administrator provides its customers with dedicated tunnels, the cloud allocation mechanism discussed in Section 3.3.1 considers the tunnels in the allocation as part of the cloud tree extending the RM and RA components to that branch of the tree. This is because the distant cloud customer network becomes part of the main cloud tree. Here is how external requests are served.

- *External Write Requests*:

  When a client wants to write data into the cloud, its UCL first requests the FES with the its (UCL) ID. The FES hashes the ID of the client and forwards the request to the respective NNS. The NNS asks the RA it chooses for the best BS for the client to write data and the rate the client request has to be served. The NNS can as well take its policies into account when it makes a decision on which BS to use. If the client is using a tunnel controlled by the cloud admin, then the RA also knows the upload and download rates to and from the UCL and the main cloud. The rate at which the UCL can write data is then the minimum of the upload rate from the UCL (to the cloud) and the download rate of the cloud (the rate at which data is written to the best BS).

  The NNS then informs the BS to contact the UCL at this rate. The UCL then starts writing to the BS after all authentication procedures. The BS sets the *receive window* of the UCL to the product of the highest level download rate it obtained from its RM and the RTT of the flow from the UCL. If the connection to the UCL is a tunnel controlled by the cloud admin, the UCL also sets its congestion window size to be the product of the highest up-link rate it obtained from its RM and its RTT. The RA can as well directly inform the UCL to send at the rate which is the minimum of the up-link rate from the UCL to the main cloud and the highest level down-link rate to the receiving BS. If the cloud admin has no control over the UCL network it simply sets the *receive window* of the flows coming (writing) to the cloud.

- *External Read Requests*:

  An external read request is when a cloud customer uses a UCL to connect to the cloud in order to read/copy data from it. To read or copy data from the cloud, the UCL first contacts the FES with its ID. The FES hashes the ID of the UCL and forward its request to the corresponding NNS. The NNS identifies the BS which has the requested data and forwards the UCL request to it (the BS). If there are multiple such BS as is usually the case due to replication, then NNS asks the RM of each BS which contains the replica. Each RM respond to the NNS with their upload rate values and the NNS chooses the BS with the highest rate or can take into account its policies. Before a new request to read comes, during replication updates, the NNS can also keep a replica leader with the highest rate to minimize the time spent asking each RM.

  At this point if the connection from the UCL to the cloud is also controlled by the cloud admin, then the NNS can also get the highest level upload rate from the RM of the requesting client. The NNS then forwards this rate value to the BS and the BS sends data to the external UCL at this rate by setting its *cwnd* to the product of the rate it obtained from the NNS and its RTT. Again this rate is the minimum of the up-link (from BS) and down-link (to UCL) rates. If the cloud has no control over the customer network, the BS simply sets its *maximum congestion window* size to the product of the highest level rate it got from its RM and the average RTT of the flow.

Serving Internal Requests

An internal cloud request is when NNS in one section of the cloud wants to send data to or copy data from another BS. For instance, a NNS may want to replicate data, or do load balancing to save energy and avoid congestion. In this case the requests may be to write data to a BS or read data from a BS.

- *Internal Write Requests*:

  Consider the case where a NNS wants to write from one BS to another BS. All BSes in a cloud share a parent (switch/router) at some level. Suppose the lowest level parent these two BS share is at level h. The NNS then contacts the RA at level h for the best BS. The RA tells the NNS the ID of the BS. The IDs of the BSs are assigned in such a way that it is possible to determine how many levels apart the BSs are until the highest level in the branch. The RA also tells the NNS at what rate the BS should send to another BS. This rate is the minimum of the up-link rate from the source to the switch at level h and down-link rate from the switch (router) to the receiver.

- *Internal Read Requests*:

  Consider the other case where the NNS wants to read data from one BS to another BS (example for replication). Suppose the lowest level parent (switch/router) the two BS share is at level h. The NNS then contacts each RM of the BS which contains the replica for their h-level rates and chooses the BS with the highest h-level rate which contains the replica. It then tells the BS with the highest h-level rate to send data to the destination BS.

Updating Rate of On-going Flows

To update the rates at which on-going flows in the cloud should send data, both the sender and the receiver have to update their windows. suppose the lowest level parent (switch/router) both the sender and receiver share is at level h. The sender sets its *cwnd* to the product of the level h upload rate it obtains from its RM and the current RTT of the flow. Besides, the receiver sets its receive window to the product of the h level download rate it obtains from its RM and the current RTT. These two *cwnd* updates in each BS are done by the RM of each BS every control interval d.

3.4 Multi-Resource QoS Aware Rate Metric

The cross-layer NCP rate metric derived in section 2.1 and 3.2 can be used with resources other than link capacity. We can show how the NCP rate can serve as a multi-resource general rate (congestion) control and resource allocation scheme. The scheme allows *unknown* number of sources to get a fair share of an X units/second amount of a certain resource ensuring a zero loss and constant buffer (zero queue length). The X units/second resource can be link capacity, spare bandwidth, processor (CPU) speed, storage per unit time, power, or any other sharable thing. The scheme allows for Quality of Service (QoS) allocation where different sources with different requirements get different shares of the resource. It can also incorporate an adaptive distributed pricing mechanism where unit prices are determined by the total demand. The details of these NCP and EDFS generalizations are presented in sections A.3, A.4 and A.5.

A typical application of the *multi-resource QoS and pricing aware* scheme is a cloud computing architecture where not only the link capacity, but also the storage and the CPU processing can be bottleneck resources as pointed out in our previous work [18, 19]. For instance if the processing (CPU) capacity or other hardware of a block server (BS) in the cloud is the bottleneck more than a link capacity, EDFS can detect it and do rate allocation and resource choice accordingly as follows.

If the rate $R_u^{BS}(t)$ at which a BS is sending data is smaller than the link rate $R_u(t)$ the BS obtains from its RM, then the BS has another local bottleneck (possibly due to other resource bottlenecks) and the RM sets its $\check{R}_0 = R_u^{BS}(t)$ and forwards this value to its RA as part of the max/min approach. This approach achieves ES (efficient sharing) in such a way that it doesn't waste the resource which the BS cannot use because it has a bottleneck other than its up-link. A hardware profiling can also be done to estimate what CPU load (rate) enables what rate of sending and receiving data packets in ways similar to what we have done in [7]. In this case the RM monitors the CPU load and translates it to link rate. If this translated link

rate is lower than the bottleneck link rate obtained from the RAs and RM, then the RM sets the sending and receiving rates to be the translated rates (due to other BS resource bottlenecks). More on this is reserved for future work.

3.5 Analytical Model on the Number of Cloudlets

The concept of cloudlets brings the cloud services closer to cloud users as cloudlets are smaller in scale. Cloudlets with some powerful storage and processing servers connected to main clouds can be scattered in a given area like WIFI hotspots. To give a full network coverage to a certain geographical area, one needs to know the number of cloudlet servers with wireless connectivity to give full coverage to all users. In [36] we have derived an analytical model to express reliability as a multi-variate function of many important parameters. This scheme can give the number of cloudlets needed for a full network coverage. The scheme can hence make sure that there is no loss of packets due to mobility or lack of connectivity.

3.6 Performance Analysis of EDFS

In this section we present some simulation results which show how the congestion control scheme of EDFS can outperform existing schemes. Other experiments on where to place data and on the advantages of using multiple name node servers (NNS) along with the front-end server (FES) in EDFS are left for the final thesis.

3.6.1 Implementation of EDFS in NS2

Referring to the EDFS architecture described by figure 3.1 in the previous section, we used the following steps in the algorithm to implement EDFS in the NS2 simulator.

- **Step1**: The RM of a node gets current rate values from TCP agents connected to the node (BS). Here interface function *between RM and TCP agents* is used.
- **Step2**: Each RM calculates the *effective number of active flows* from these rates using equation 3.3 and sends it to the RA agent using interface function *between RM and RA agents*. Each RA calculates the effective number of active flows from these values and sends it to its parent RA using an interface function *between child RA and parent RA* agents.
- **Step3**: Each RM and RA agents contact the Queue (DropTail) agent using an interface function *between RM or RA and DropTail* agents to get the current queue size and total arrival rate (for simplified EDFS) necessary for the calculation of the rate metric. Each RM and RA then calculates its rate metric using equation 3.2.
- **Step4**: Each RM and RA sets its *upward rate allocation* to the minimum of its own (obtained using equation 3.2) and the highest rate it got from its children. Each RA forwards this to its parent RA and the parent RA does the same. When this process gets to the top level RA, each RA and RM has its level's best (highest) rate value with the ID (address) of the corresponding BS. The top k best BS can also be selected instead of just 1 best BS. The NNS can then use this information to make h-level decision on *where to store data*; that is by contacting the h-level RA whenever a request to store data in the cloud comes.
- **Step5**: Each RA sends its rate value obtained using equation 3.2 along with its level number to its children. Each child RA also forwards to its children the minimum of its rate value obtained using equation 3.2 and each h-level rates it obtained from its parent rate. When this process reaches a RM, the RM has all level rates each of which are the minimum of the rates up to each level. This step helps the NNS for instance when a client wants *to read data from a specific BS*. In this case the NNS contacts the RM of the BS to know the best rate at which data can be transfered to and from the BS. When a BS wants to send data to another BS which shares the lowest level switch (RA) at level h, then the sender sets its *cwnd* to the product of its h-level up-link rate and the RTT of its flow. The receiver also sets its receive window to the product of its h-level down-link rate and the RTT of the flow. This is done by the another common interface function between the RM and TCP.

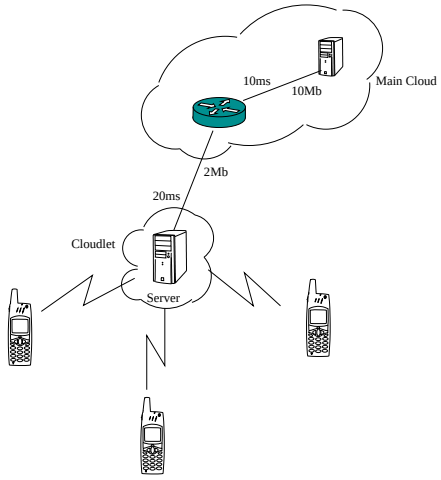
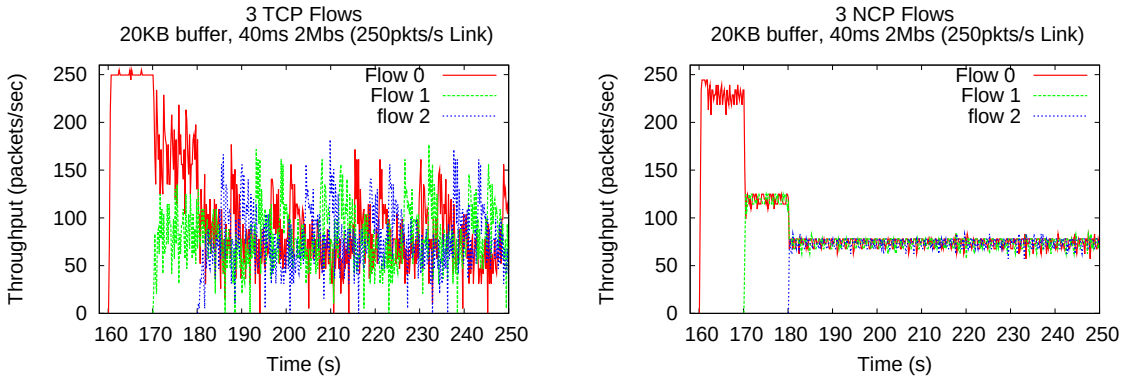


Figure 3.2: Simple cloudlet simulation topology



(a) Throughput versus Time

(b) Throughput versus Time

Figure 3.3: Comparison of EDFS (NCP based) and GFS (TCP based) with respect to throughput.

3.6.2 Congestion Control Analysis: HDFS (TCP-based) vs EDFS (NCP-based)

To analyze the performance of EDFS against the well known HDFS (GFS) we first used a simple topology for a scenario where (mobile) nodes download data from the cloud. Then we experimented with a more complex topology where (mobile) nodes upload data to the cloud.

Simple Simulation Topology

Figure 3.2 shows the simple topology with three fixed mobile devices with a fixed cloudlet.

As shown in figures 3.3(a) and 3.3(b) the NCP based (EDFS) approach achieves the fair share of the bandwidth faster than the TCP based approach. Besides, the NCP-based approach gives more smooth throughput.

Larger Topology

The larger network topology we used to validate the performance of NCP inside EDFS is given in figure 3.4. In this topology the mobility area is 637m x 637m rectangular area ($l = w = 637$). The transmission range $t = 150m$. To get coverage of the rectangular area with 100% reliability, from our previous work we need m

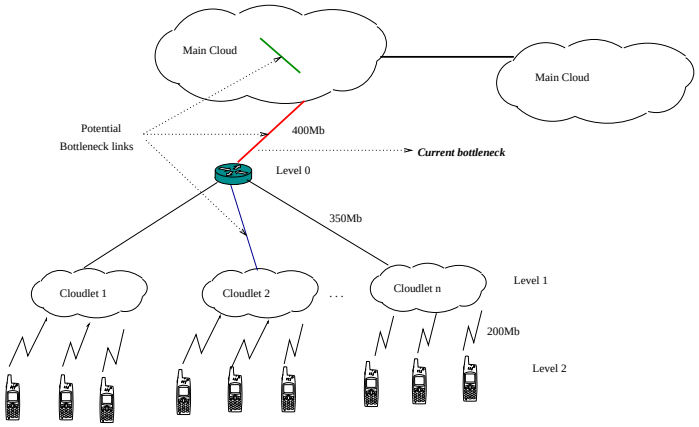


Figure 3.4: Larger cloudlet simulation topology

servers given by

$$m = \left\lceil \frac{\sqrt{2}l}{2t} \right\rceil \times \left\lceil \frac{\sqrt{2}w}{2t} \right\rceil = 9. \tag{3.6}$$

10 Mobile nodes are connected to each cloudlet. The number of TCP sources which start from each mobile node follows a Poisson distribution with Pareto file size. In our first experiment of this complex topology the link connecting the cloudlets to the main cloud is a bottleneck. In this simulation instead of the wireless nodes we used high bandwidth wired end-hosts to observe the performance of the cloudlet architecture when the link to the main cloud is the bottleneck.

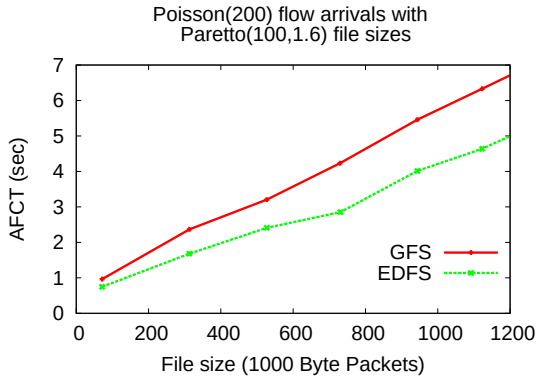
As shown in figures 3.5(a), 3.5(b) and 3.5(c) where file size is in 1000 byte packets, the NCP-based approach (EDFS) gives smaller delay when compared with the TCP-based approach (GFS). Even with bigger size flows, the NCP-based (EDFS) architecture outperforms the TCP based (GFS) approach. The performance gain of the NCP based architecture is even better with bigger file sizes as shown in 3.5(c).

3.7 Summary

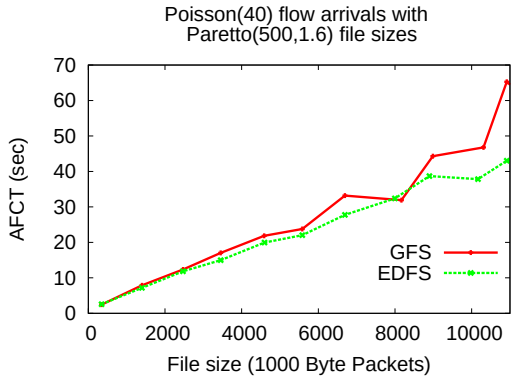
In this chapter we presented the design of an efficient distributed file system (EDFS) for large scale systems such as cloud computing. Current large scale distributed file systems such as the Google File System (GFS) and its derivate the Hadoop File System (HDFS) rely on a single name node server (NNS) to manage metadata information of all chunks stored in all block (chunk) servers (BS) in the cloud. This design can make GFS and HDFS bottlenecked at the single NNS. The design of EDFS solves this problem by introducing a light weight front end server (FES) which forwards requests to multiple NNS.

The EDFS architecture also uses a scheme similar to the Network Control Protocol (NCP) discussed in the previous chapter to decide *where in the cloud (distributed system)* to store data and *at what rate to transmit* data. This design enables EDFS to efficiently balance load among all data and name node servers.

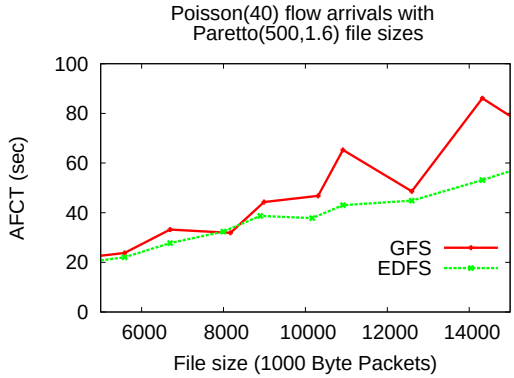
We have implemented the EDFS scheme in NS2 which is a widely used network simulator. We then compared how its NCP-like component enables EDFS to outperform GFS and HDFS which rely on TCP and hence decrease average file completion time (AFCT). The resource monitor (RM) and resource allocator (RA) components of EDFS also allow EDFS to be implemented without the need to change network switches, routes and the TCP/IP packet header format.



(a) AFCT versus file size



(b) AFCT versus file size



(c) AFCT versus file size

Figure 3.5: Comparison of EDFS and GFS with respect to AFCT.

Chapter 4

Research Plan

4.1 Current Progress

We have designed the cross-layer congestion control and routing network control protocol (NCP) as described in chapter 2. We have evaluated the congestion control component of NCP against another clean slate congestion control protocol using the NS2 simulator where we implemented NCP.

We have also designed an efficient distributed file system (EDFS) by extending the NCP concepts as discussed in chapter 3. We have implemented EDFS in the NS2 simulator and made simulation analysis to evaluate the congestion control component of EDFS against the congestion control component of other well known distributed file system architecture.

4.2 Next Steps

The next steps of this PhD research can be summarized as follows:

- **Discussion on Cross-layering**: We plan to discuss the cross-layer (routing) component of NCP as shown in [17].
- **Enabling NCP Implementation in Current Networks**: We plan to discuss the BRTP [21], NetMap [14] and eNCP [23] concepts which can enable NCP to be easily implemented in current networks.
- **Experiments EDFS on Where to Place Data**: So far we have performed experiments showing that the congestion control component of EDFS outperforms the congestion control component of GFS. We also plan to show with simulation results the benefits of the EDFS component which deals with *where in the cloud to store data* and how to route data in the cloud where there are multiple possible paths.
- **More EDFS Results on Cloudlet mobility**: We will also conduct EDFS simulation experiments when the cloudlets are mobile and where data is stored in the cloudlets until the cloudlets get connected to the main cloud.
- **Multi-Resource EDFS**: We plan to discuss how the EDFS design takes into account other resource bottlenecks such as the storage and processing.
- **More on Stability Analysis**: We plan to extend the current stability analysis of NCP to include a wider range of values for the stability parameters. We also plan to discuss stability analysis of NCP and hence EDFS using *some fixed point theorems.*

4.3 Research Time line

The current time-line of the PhD thesis is as follows. I plan to finish all main experiments of the topics discussed in section 4.2 above by April 28, 2012. From May 02, 2012 to July 28 I plan to finish the write-up while also dealing with minor experiments.

Appendix A

Extra Sections on NCP and EDFS

In this chapter we present some derivations and extensions of NCP and EDFS. In sections A.1 and A.2 we show how other major clean slate protocols can be considered as subsets of NCP. In the remaining sections A.3, A.4 and A.5 we show how the concept of NCP and EDFS can be extended for resource bottlenecks other than the link capacity. Finally in section A.6 we give a brief summary of the apendix.

A.1 Does RCP really closely emulate processor sharing?

In [10] and other similar works, RCP is reported to closely emulate processor sharing (PS). In the simulation setup used to evaluate the performance of RCP flow 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 almost 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.

$$f_{rate} = \frac{\rho C}{f_{size}} = \frac{\Lambda}{f_{size}}. \tag{A.1}$$

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.

Therefore such 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 [10] as it under or over estimates the number of active flows by which the link capacity has to be divided. 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). 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.

A.2 General Cases when RCP works well: Derivation using NCP

Putting Equation 2.17 into Equation 2.1

$$N(t) = \frac{\left(\alpha C - \alpha y(t) - \beta \frac{q(t)}{d}\right) \sum_{j=1}^{L_i} (1/w_j)}{\alpha C - \beta \frac{q(t)}{d} - R(t-d) \sum_{j=1}^{L_i} (1/w_j)}. \tag{A.2}$$

Therefore the specific scenario where RCP works well is when the approximate value of $N(t)$ which is $C/R(t-d)$ equals the exact value given by Equation A.2 above. That is when

$$\frac{C}{R(t-d)} = \frac{\left(\alpha C - \alpha y(t) - \beta\frac{q(t)}{d}\right)\sum_{j=1}^{L_i}(1/w_j)}{\alpha C - \beta\frac{q(t)}{d} - R(t-d)\sum_{j=1}^{L_i}(1/w_j)}. \tag{A.3}$$

This implies that

$$\alpha C^2 - \left(\beta\frac{q(t)}{d} + (1+\alpha)y(t)\right)C + y(t)\left(\alpha y(t) + \beta\frac{q(t)}{d}\right) = 0. \tag{A.4}$$

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 $\alpha = 0.1$, $\beta = 1.0$, $q(t)/d = 5$, $C = 10$ we get $y(t) = 25.62$.

For all values which do not satisfy Equation A.4 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.

Deriving XCP from NCP

The rate allocation scheme of an XCP-like algorithm can be derived from the NCP ideas as follows. The main idea of XCP is to divide the spare bandwidth $S = C - \Lambda - q(t)/d$ 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

$$\sum_{j}^{L}\frac{\Delta R}{R_j} = dS. \tag{A.5}$$

This implies that

$$\Delta R = \frac{dS}{\sum_j^L \frac{1}{R_j}}. \tag{A.6}$$

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

$$\begin{aligned} R_i^{new} &= R_i + \Delta R \\ &= R_i - \frac{d\Lambda}{\sum_j^L \frac{1}{R_j}} + \frac{dC - q(t)}{\sum_j^L \frac{1}{R_j}} \\ &= R_i - \frac{d\Lambda}{\sum_j^L \frac{1}{R_j}} + R(t)^{ncp} \end{aligned} \tag{A.7}$$

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.

The above rate representation of some general variant of NCP can now be modified to achieve different objectives. For instance if $R = \frac{d\Lambda}{\sum_j^L \frac{1}{R_j}}$ one can multiply new feedback value $\Phi = R(t)^{ncp} - \frac{d\Lambda}{\sum_j^L \frac{1}{R_j}}$ in equation A.7 with R_i/R if it is needed to keep both the increase and decrease of flow sending rates proportional to its current rate or with R/R_i if one wants both the increase and decrease in a flow i 's sending rates inversely proportional to flow i's current sending rate.

In scheme used in XCP [37] increases the throughput of all flows equally if $\Phi > 0$ and decreases the throughput of a flow proportional to its current throughput if $\Phi < 0$. Hence to do a scheme similar to what XCP does using NCP, multiply Φ with R_i/R if $\Phi < 0$ and keep equation A.7 unchanged if $\Phi > 0$. The bandwidth shuffling concept used in XCP where the simultaneously allocate and deallocate bandwidth to flows if $\Phi = 0$ can also be used here.

So from the analysis in the previous sections we can see that both XCP and RCP can be subsets of NCP.

A.3 Basic Formulation of Multi Resource NCP

Suppose that each of an unknown number of sources i wants to use the common resource at a rate of R_i units/second. Without loss of generality let the units be packets. If the common resource receives a total of L packets during the control interval d from the unknown number of sources, then the fair share rate R of each of the unknown number of sources which can be put in the packets or other probe packets is derived as follows.

The sum of the per packet j fair share rate of all the packets which arrive to the resource from all unknown number of sources shouldt exceed the total resource capacity X. That is

$$\sum_{j}^{L} \frac{R}{R_j d} = X \tag{A.8}$$

which implies that

$$R = \frac{Xd}{\sum_{k}^{L} \frac{1}{R_j}}. \tag{A.9}$$

In the simplest case of this general scheme where all sources send at the same rate R_k which was given to them in the previous interval (round k) or if the current share of each source is approximated by the previous share R_k then the share of each resource for the next round $k+1$ can be given by

$$R_{k+1} = \frac{X}{\frac{L}{R_k d}} = \frac{X}{\frac{\Lambda}{R_k}} \tag{A.10}$$

where $\Lambda = \frac{L}{d}$.

This approximation of the equal share of each source doesnt require packets to carry any current rate information of their respective sources and is much simpler to implement.

The resource amount X can be given in different forms. If the resource to share is a link, X in its simplest form can be replaced with the link capacity C or spare bandwidth $S = C - \Lambda$. Using some additive queue control terms the capacity X can also be replaced with $\alpha C - \beta Q/d$ where α and β are control parameters chosen for stability and performance and Q is the queue length as used in NCP, RCP [10] and XCP [37]. The value of X can also be given by $\alpha\Lambda + F$ where the aggregate feedback rate F is as shown in [1]. Using multiplicative queue control schemes X can also be replaced with the effective bandwidth $C \times f(Q)$ where $f(Q)$ can be the hyperbolic or other queue control functions given at [49]. Other similar resource expressions can replace X in equations A.9 and A.10.

A.4 Multi-resource rate for QoS

The multi-resource rate can also give QoS solutions. If packet j of the L packets which arrive to the resource during the control interval d has a quality of service requirement p_j then equation A.8 becomes

$$\sum_{j}^{L} \frac{p_j R}{R_j d} = X \tag{A.11}$$

which implies that

$$R = \frac{Xd}{\sum_{j}^{L} \frac{p_j}{R_j}}. \tag{A.12}$$

Then a source with a QoS requirement p_j takes the share $p_j R$.

A.5 Multi-Resource Rate Pricing Scheme

An adaptive pricing and incentive scheme can also be introduced into the multi-resource rate metric discussed in the above section A.4. If the unit price of a resource at round k is $\wp_k$, then each resource wants to maximize

the value

$$V_{k+1} = \frac{R_{k+1}}{\wp_k} \tag{A.13}$$

and the price for the next round $k+1$ becomes

$$\wp_{k+1} = \frac{R_k}{V_{k+1}} = \frac{R_k \wp_k}{R_{k+1}}. \tag{A.14}$$

So if the rate R_{k+1} at round $k+1$ obtained by equation A.12 is higher than the rate R_k of the previous round obtained by the same equation but in the previous round k, then there is more resource to share and hence cheaper unit price and vice-versa. More detailed analysis and discussion of this concept is reserved for future work.

A.6 Summary

In this appendix we have shown how RCP does not really emulate processor sharing (PS). We have in Chapter 2 shown that NCP can emulate PS. We have also shown how RCP and XCP which are the two other major clean slate protocols can be derived using the NCP scheme. These other protocols can also be treated as subsets of NCP. In this appendix, we have also discussed how NCP and hence EDFS can be extended to deal with multi-resource bottlenecks. We further described how this multi-resource NCP can provide QoS and pricing schemes.

Bibliography

[1] Abrantes, F., and Ricardo, M. Xcp for shared-access multi-rate media. *SIGCOMM Comput. Commun. Rev. 36* (July 2006), 27–38.

[2] Akella, A., Chawla, S., Kannan, A., and Seshan, S. Scaling properties of the internet graph. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing* (New York, NY, USA, 2003), PODC '03, ACM, pp. 337–346.

[3] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev. 40* (Aug. 2010), 63–74.

[4] Anderson, N., and Cerf, V. One quarter of all computers part of a botnet. `http://arstechnica.com/news.ars/post/20070125-8707.html`, 2007.

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

[6] Bennett, J. C. R., and Zhang, H. Hierarchical packet fair queueing algorithms. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 1996), ACM, pp. 143–156.

[7] Cao, Q., Fesehaye, D., Pham, N., Sarwar, Y., and Abdelzaher, T. Virtual battery: An energy reserve abstraction for embedded sensor networks. In *Proceedings of the 2008 Real-Time Systems Symposium* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 123–133.

[8] Dawkins, S., Montenegro, G., Kojo, M., Magret, V., and Vaidya, N. End-to-end performance implications of links with errors. RFC Editor.

[9] Dixon, Colin, Anderson, Thomas, Krishnamurthy, and Arvind. Phalanx: withstanding multimillion-node botnets. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), USENIX Association, pp. 45–58.

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

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

[12] Dukkipati, N., McKeown, N., and Fraser, A. G. RCP-AC: Congestion Control to Make Flows Complete Quickly in Any Environment. In *INFOCOM* (2006).

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

[14] Fesehaye, D. Dynamic Mapping of an AS Network into A Smaller Network of Border Routers. Technical report, University of Illinois at Urbana-Champaign (UIUC), 12 2010.

[15] Fesehaye, D. GreenEDFS: Saving Energy in the Clouds Using an Efficient Distributed File System . Technical report, University of Illinois at Urbana-Champaign (UIUC), 012 2010.

[16] Fesehaye, D., Ahmed, S., Pongthawornkamol, T., Nahrstedt, K., and Wang, G. Reliability Trade-off Analysis of Deadline-Sensitive Wireless Messaging Systems. Technical report, University of Illinois at Urbana-Champaign (UIUC), 03 2011.

[17] Fesehaye, D., Gupta, I., and Nahrstedt, K. A Cross-layer Routing and Congestion Control for Distributed Systems. Technical report, University of Illinois at Urbana-Champaign (UIUC), Nov. 2008.

[18] Fesehaye, D., Malik, R., and Nahrstedt, K. EDFS: a semi-centralized efficient distributed file system. In *Middleware (Companion)* (2009), p. 28.

[19] Fesehaye, D., Malik, R., and Nahrstedt, K. A Scalable Distributed File System for Cloud Computing. Technical report, University of Illinois at Urbana-Champaign (UIUC), 03 2010.

[20] Fesehaye, D., Malik, R., and Nahrstedt, K. Efficient Distributed File System (EDFS). Technical report slides, University of Illinois at Urbana-Champaign (UIUC), 07 2010.

[21] Fesehaye, D., and Nahrstedt, K. BRTP: Border Routing & Transport Protocol. Technical report, University of Illinois at Urbana-Champaign (UIUC), 08 2010.

[22] Fesehaye, D., and Nahrstedt, K. PAS: A Packet Accounting System to Limit the Effects of DoS & DDoS. Technical report, University of Illinois at Urbana-Champaign (UIUC), 07 2010.

[23] Fesehaye, D., and Nahrstedt, K. Implementation of the Network Control Protocol using ECN Bits (eNCP). Technical report, University of Illinois at Urbana-Champaign (UIUC), 05 2011.

[24] Fesehaye, D., Nahrstedt, K., and Caesar, M. A Network Congestion control Protocol (NCP). In *CS/UIUC Technical Report* (Urbana, IL, USA, 2010).

[25] Finn, P. Cyber assaults on estonia typify a new battle tactic. `http://www.washingtonpost.com/wp-dyn/content/article/2007/05/18/AR2007051802122`, 2007.

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

[27] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev. 37* (Oct. 2003), 29–43.

[28] Gu, G., Perdisci, R., Zhang, J., and Lee, W. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)* (2008).

[29] Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)* (August 2007).

[30] Gu, G., Zhang, J., and Lee, W. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)* (February 2008).

[31] Gunes, H., M., Bilir, Sevcan, Sarac, Kamil, Korkmaz, and Turgay. A measurement study on overhead distribution of value-added internet services. *Comput. Netw. 51*, 14 (2007), 4153–4173.

[32] Henderson, T. R., Sahouria, E., Mccanne, S., Katz, R. H., and Katz, Y. H. On improving the fairness of tcp congestion avoidance. In *IEEE Globecomm conference* (1997), pp. 539–544.

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

[34] Kandula, S., Katabi, D., Davie, B., and Charny, A. Walking the tightrope: responsive yet stable traffic engineering. *SIGCOMM Comput. Commun. Rev. 35* (August 2005), 253–264.

[35] Kar, K., Kodialam, M., Lakshman, T. V., and Member, S. Minimum interference routing of bandwidth guaranteed tunnels with mpls traffic engineering application. *IEEE Journal on Selected Areas in Communications* (2000), 2579.

[36] Kassa, D. F., Nahrstedt, K., and Wang, G. Analytical models of short-message reliability in mobile wireless networks. In *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems* (New York, NY, USA, 2011), MSWiM '11, ACM, pp. 369–376.

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

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

[39] Labovitz, C. 2008 worldwide infrastructure security report. `http://asert.arbornetworks.com/2008/11/2008-worldwide-infrastructure-security-report/`, 2008.

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

[41] Lu, Z., and Zhang, S. Stability analysis of xcp congestion control systems. In *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE* (april 2009), pp. 1 –6.

[42] Mahimkar, A., Dange, J., Shmatikov, V., Vin, H., and Y.Zhang. dfence: Transparent network based denial of service mitigation. In *NSDI'07: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2007), USENIX Association.

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

[44] Moy, J. T. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[45] Powell, R. Cotendo unveils cloudlets, more telco alliances coming? `http://www.telecomramblings.com/2011/04/cotendo-unveils-cloudlets-more-telco-alliances-coming/`, 2011.

[46] Rosen, E., Viswanathan, A., and Callon, R. Multiprotocol label switching architecture. RFC Editor.

[47] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (oct.-dec. 2009), 14 –23.

[48] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (Washington, DC, USA, 2010), MSST '10, IEEE Computer Society, pp. 1–10.

[49] Vandalore, B., Jain, R., Goyal, R., and Fahmy, S. Design and analysis of queue control functions for explicit rate switch schemes. In *ICCCN* (1998), pp. 780–786.

[50] Wang, J., Yang, Y., Xiao, L., and Nahrstedt, K. Edge-based traffic engineering for ospf networks. *Comput. Netw. 48* (July 2005), 605–625.

[51] Yaar, A., Perrig, A., and Song, D. X. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *IEEE Symposium on Security and Privacy* (2004), pp. 130–.

[52] Yang, Xiaowei, Wetherall, David, Anderson, and Thomas. A DoS-limiting network architecture. *SIGCOMM Comput. Commun. Rev. 35*, 4 (2005), 241–252.