Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

Ensuring Transport Fairness with Smart Networks

Semester Thesis

Author: Long He

Tutor: Alexander Dietmüller, Maria Apostolaki

Supervisor: Prof. Dr. Laurent Vanbever

March 2020 to July 2020

Abstract

In today's Internet, tremendous amounts of network flow with many different protocols share all the network resources, and unfairness between protocols happens due to their different property. Currently, the network itself is mostly treated as a dumb medium that provides all its resources first-come-first-serve, and it is up to the hosts to ensure fairness. However, hosts fail to ensure fairness as more and more new protocols emerging, and ensuring fairness between protocols puts great a burden on designing the new protocol and adopting them. I propose a new flow measurement and queue management algorithm to measure unfairness in-network and ensure fairness in-network, thus network devices can take over the task of ensuring fairness. I implement my algorithm on the ns-3 simulator, and the result shows my method can measure fairness with accuracy higher than 90%; and my method is able to improve fairness in a large congestion control algorithm sets.

Contents

Chapter 1

Introduction

1.1 Motivation

One of the fundamental challenges of the communication network is fairness. More precisely, how to fairly allocate network resources to different network flows.

1.1.1 The Current on-host approach fails to ensure fairness among different protocols

Currently, the network itself is mostly treated as a dumb medium that provides all its resources first-come-first-serve, and it is up to the hosts to ensure fairness. However, for each host, it is naturally challenging to determine its fair share, because how much its fair share should be, is highly dependent on other hosts' flows, and the whole network, and hosts usually do not have such information. Thus currently, hosts usually try to determine their fair share indirectly with the help of protocols. Protocols will listen to signals in the network (e.g. packet loss, delay, etc.) and act to those signals, and it is carefully designed such that fairness is ensured for many concurrent flows using the same protocol. However, the problem is, as soon as flows are using different protocols, fairness cannot be ensured anymore, because different protocols listen to different signals and react differently to the signals.

In this paper, I focus on TCP protocols and TCP variants with different congestion control algorithms (CCA). Different CCA usually reacts differently to packet loss or/and delay, which causes unfairness between different TCP variants. As discussed above, TCP can ensure fairness if flows are using the same TCP variant. For example, TCP usually increases the sending rate of flows until packets are dropped (indicating resource exhaustion), at which point it reduces the sending rate. The concrete changes in sending rate are carefully designed such that multiple concurrent flows converge to a fair steady state. If all hosts in the network were to use the exact same TCP protocol, this would be feasible. However, it is not the case in today's Internet. Modern networks carry traffic from many different variants of TCP alone, among other protocols. Even if there is only a small difference between two TCP variants, it may break the convergence towards a fair steady state.

A concrete example is TCP NewReno and TCP Vegas. NewReno gradually increases its sending rate until packets are drop (indicating congestion). Differently, Vegas measures delay (RTT), and decrease rates if RTT increases substantially (also indicating congestion). In short, NewReno will try to make the buffer full, and only decreases rate when the buffer is full (where packets are dropped). However, Vegas will try to keep a low delay, also a short buffer. For example, if the

buffer is large in the bottleneck link, NewReno will harm Vegas in the competition between them, and unfairness happens.

1.1.2 The Current on-host approach slows down developing new protocols.

Trying to let hosts ensure fairness is a fundamental design flaw that unnecessarily slows down the design of new and improved transport protocols. On the one hand, having to consider 'being fair' to other protocols puts an unnecessary burden on protocol designers. Currently, when developing new protocols, designers not only need to consider the new protocol its own performance but also need to carefully evaluate how well it reacts with existing protocols. For example, it is possible that the new protocol itself performs well when it is alone, but doesn't reach the same performance when competing with other protocols; or the new protocol gets more than the fair share and starve existing protocols. If such happens, they have to modify the new protocol, until it gets along with other protocols. However, there are many different protocols in the network. To ensure fairness, designers usually have to exam a lot of different protocols, which is of course a hard and even endless process.

On the other hand, it makes the adoption of new protocols difficult. Designers usually cannot fully exam the coexisting of a new protocol with other protocols at all circumstances. Thus many problems can happen after adopting the new protocols to the network. For example, the new protocols may harm some existing protocols a lot, and affect user experience and the whole network.

1.1.3 The network has the potential and advantages to ensure fairness.

Thus, I want network devices to take over this task. Network devices naturally have advantages compared to hosts for the task of ensuring fairness. First of all, network devices can see all the network flows from different hosts, such knowledge give us the probability to determine each flow's fair share directly. Secondly, the concrete fair share typically does not depend on what transport protocols the flows are using. This means the mechanism I designed on the network device is applicable to all different TCP variants, even to new variants in the future. Finally, in network devices, resource allocation revolves nearly exclusively around the device buffer, which is typically a set of queues. I can control the in and output to these queues, e.g. how many packets per second to enqueue and dequeue, to schedule different flows and ensure fairness.

Of course, there are many challenges to let the network take over the task of ensuring fairness. A fundamental challenge is the type of protocol is usually invisible to network, which means current network does not have knowledge of what exactly protocol (e.g. what kind of CCA) a flow is using. But in this paper, I didn't solve this problem. I just assume that suppose the network can distinguish different protocols, and focus on how to ensure fairness between protocols in the network.

1.2 Task and goals

I have three main tasks in this project: *(i)* define and quantify unfairness; *(ii)* measure unfairness in the network devices; *(iii)* prioritize flows based on my measurements.

To begin with, a first key challenge is to find a suitable definition of unfairness, and how can I calculate this unfairness, suppose given all information on the flows I need in the network.

After that, I need to design my method to measure unfairness inside network devices with limited information. In the network device, it is usually not possible to record the full information of different flows, because of limited computation and memory resources. But I can take advantage

of the information in the network device, e.g. the queue length, packets rate, and throughput, etc., to help to measure unfairness. And I also need to compare my measurement with the "true value" explained above, to make sure my measurement is accurate.

Finally, based on the unfairness I measured in the network device, I can implement my queue management algorithm to prioritize different flows. Still, I need to check whether my method improves fairness and how much it can improve.

1.3 Overview

Section 2 provides background information to understand this paper and related work. Section 3 presents a detailed design of algorithm and discussed the reason for such design. Section 4 gives the evaluation result of fairness measurement and fairness improvement. Section 5 discusses the outlook of the work, and section 6 gives a finally short summary.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 TCP Protocols

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network. TCP is a connection-oriented protocol. A connection (or a flow) is defined by its 5-tuple attributes, including source and destination IP, source and destination port and the protocol field.

TCP uses network congestion-avoidance algorithms to avoid network congestion. And congestion control is largely a function of an internet host, not the network itself. In other words, it is up to the host to determine its packets rate. A typical congestion control algorithm (CCA), e.g. TCP Reno, uses a state variable congestion window (`cwnd`) to determine a packets rate upper limit, and the algorithm itself updates `cwnd` through a TCP connection.

Additively increase/multiplicatively decrease (AIMD) is the most typical scheme used in CCA, and it is proved that for concurrent TCP flows using the exact same CCA, they will converge to a fair steady state[2]. For example, TCP usually additively increase `cwnd` until packets are dropped, at which point it multiplicatively decrease `cwnd`. In short, if in the beginning, some flows have higher packets rate than others when network resources exhausted and packets are dropped, those flows will also decrease more than others, thus finally the network reaches a fair steady state.

If all hosts in the network were to use the exact same TCP CCA, it would be feasible to ensure fairness. However, there are many different CCAs using different signals and algorithms in today's internet. For example today there are three main different types CCA, loss-based CCA, delay-based CCA and hybrid CCA[5]. Interaction between these different CCAs is investigated a lot recently. I will show some of the results in the following Related Work section.

2.2 Related Work

Today's internet is filled with tremendous heterogeneous protocols. Even if I only focus on TCP flows, there are still many TCP variants, and the percentage of each is ever-changing[4] Recent studies show that combining different CCAs and active queue management (AQM) evolved in the past few years results in poor fairness[3]. However, all those protocols and CCAs are actually carefully examined to ensure fairness and not harm existing protocols when they are designed. For example, BBR is a new congestion control algorithm deployed for Chromium QUIC and the Linux

kernel. It has rapidly become a major player in Internet congestion control. In BBR's original paper, authors show that a BBR flow is fair to the other TCP flow. But a recent study shows significant unfairness when BBR competes with traditional TCP.[7] It is indeed fair when there is only one BBR flow and one TCP flow, but as soon as there are multiple BBR/TCP flows, it is not fair anymore. In a word, it is very hard to design a new protocol that is fair to exist protocols, because the real network environment is way too complicated. As achieving fairness between new CCAs and deployed CCAs is hard, instead, there are researches focuses on limiting the harm caused by new CCAs[6].

There are also attempts to measure behaviors of CCAs in the network and dynamically changes the buffer.[1] It works as an AQM scheduler based on fair queuing, which adapts the buffer size depending on the needs of each flow without requiring active participation from the endpoint. It improves the performance of different CCAs but didn't solve unfairness between protocols.

Chapter 3

Design

Suppose there are many TCP flows with different TCP variants in the network, if I have all the information in the network, then how can I determine whether a type of TCP variant is unfair to other flows? For example I have m TCP variants, a naive solution is to quantify each protocol $V = [v_1, v_2, ..., v_m]$, then use a classical fairness metric, e.g. Jain's fairness index: $J(v_1, v_2, ..., v_m) = \frac{(\sum_{i=1}^{m} v_i)^2}{m \sum_{i=1}^{m} v_i^2}$. However, there is a challenge on how to quantify each protocol. For example, if I just simply measure each TCP variant per-flow average throughput, then it may not truly reflect unfairness. It is possible that a flow has much lower throughput than another concurrent flow, while the two concurrent flows are still fair to each other. For example, the lower throughput flow is from an old-fashioned mobile device, while the higher throughput flow is from a desktop with a brand-new network interface card. The difference in their rates is not from the competition of the two flows, thus they are still fair to each other, even if there is a huge gap between their rates.

3.1 Define Fairness

Thus I want to measure the "slowdown" caused by competition with other protocols, and if there is no protocol is affect worse, then fairness is ensured. More formally, I extend my fairness measurement to the "slowdown" scenario. I define $V^{\theta} = [v_1^{\theta}, v_2^{\theta}, ..., v_m^{\theta}]$ as the metrics for each protocol, when they do not compete with each other. And $V = [v_1, v_2, ..., v_m]$ as metrics when all the m protocols are competing with each other. Then $J(V, V^{\theta}) = \frac{(\sum_{i=1}^{m} v_i/v_i^{\theta})^2}{m \sum_{i=1}^{m} v_i^2/v_i^{\theta 2}}$.

3.2 RRMQS: Round Robin Measuring Queues Set

In practice I randomly pick some flows as samples, to find V^{θ}, the metrics when protocols are not competing with each other. I use a dedicated round-robin measuring queues set (RRMQS) to realize such function. The RRMQS will have m small queues (recall that I have m TCP variants in the network), and these m queues use round-robin dequeue scheme. Each queue in the RRMQS only serves one fixed protocol, thus protocols are not competing with each other in the RRMQS. I also make sure that each queue in RRMQS has the same flow capacity and packets capacity, meaning in each queue there will be at most C_f flows and at most C_p packets at any time. Usually in the network, the number of flows for different TCP variants is large ($>> 1$), thus for a small C_f (e.g. $C_f < 10$), the number of flows in each queue will nearly always be C_f. As in each queue of RRMQS, they has the same number of flows inside, and they use a round-robin dequeue scheme,

fairness for TCP variants in RRMQS is also ensured.

Aside from RRMQS, my network devices will also have another large main queue to serve all the other flows. The part of flows which using the main queue may suffer unfairness, but flows using RRMQS is an exactly fair result. Thus I can compare flows using RRMQS and main queue, to decide whether unfairness happens in the main queue.

my queue discipline uses a simple way to enqueue and dequeue. When a new flow coming and the corresponding queue in RRMQS is available, that flow will use RRMQS; otherwise it will use the main queue. When dequeuing a packet, there is a fixed probability to dequeue RRMQS or main queue, so that they share the resources with a fixed ratio. If dequeue RRMQS, it will use a round-robin dequeue, which means all the m queue will dequeue in turn. I will provide detail of the algorithms in the following.

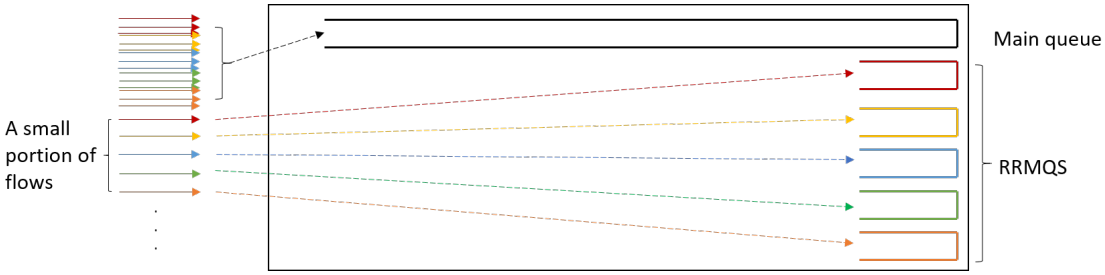

Figure 3.1: Illustrate main queue and RRMQS, different color represent different TCP variants.

Finally, the main queue and RRMQS consist of the queue discipline. Formally, there are $m+1$ queues, with m queues each size C_p packets, and one main queue with size C_{pm}, and $C_{pm} >> C_p$. Now I describe how to initialize, enqueue and dequeue my queue discipline.

Initialize As described above, I need m queues in RRMQS and 1 queue for the main queue. In addition, I need some memory (e.g. a vector) to record which flows are in RRMQS and a integer for the round-robin-dequeue function for RRMQS. I show the pseudo-code in Algorithm 1.

Algorithm 1 Initialize Queue Discipline

Input: number of different TCP variants m;
1: Initialize RRMQS with m empty queue;
2: Initialize main queue with a empty queue;
3: $RRMQS\_index = 0$; /*index used for round robin dequeue*/
4: Initialize a empty vector $flow\_vector$ to record 5-tuple of flows.

Enqueue: For simplicity, when a new flow coming and the corresponding queue in RRMQS is available, this flow should use RRMQS; otherwise if RRMQS is not available, use the main queue instead. I show the pseudo-code in Algorithm 2.

Dequeue: When doing dequeue, if both main queue and RRMQS are not empty, then randomly choose the main queue or RRMQS with fixed probability; otherwise if only the main queue/RRMQS is not empty, then dequeue main queue/RRMQS. When dequeue RRMQS, I use a round-robin scheme to ensuring fairness between queues in RRMQS. I show the pseudo-code in Algorithm 3.

3.3 Flow Meter: represent and quantity unfairness between CCAs

Once I separate some flows as samples and schedule them in RRMQS as a fair baseline, I can analyze flows in the main queue and in RRMQS separately. Furthermore, after comparing the

Algorithm 2 Enqueue

Input: packet p;
Output: If successfully enqueue (true/false);

```
get 5-tuple, TCP flags and TCP variants type i from p;
if p is from a new flow then
    if C_f[i]    1 then /*RRMQS is available*/
        record 5-tuple in flow_vector;
        ret = RRMQS[i].enqueue(p);
    else/*RRMQS isn't available*/
        ret = main_queue.enqueue(p);
    end if
else
    if 5-tuple is in flow_vector then
        ret = RRMQS[i].enqueue(p);
    else
        ret = main_queue.enqueue(p);
    end if
end if
return ret;
```

statistical result for the main queue and RRMQS, I can determine the unfairness in the main queue. I can use a so-called "flow meter" to analyze flows statistically. The basic idea is I use a flow meter to measure main queue, and get m values for m TCP variants, $V = [v_1, v_2, ..., v_m]$. And I use another flow meter to measure RRMQS, and get $V^{\theta} = [v_1^{\theta}, v_2^{\theta}, ..., v_m^{\theta}]$. I somehow compare V and V^{θ} to calculate each TCP variants' fairness index $I = [i_1, i_2, ..., i_m]$. And the choice of what I count and analyze (e.g. throughput, packets rate, flow completion time...) will has a great impact on the accuracy of fairness measurement.

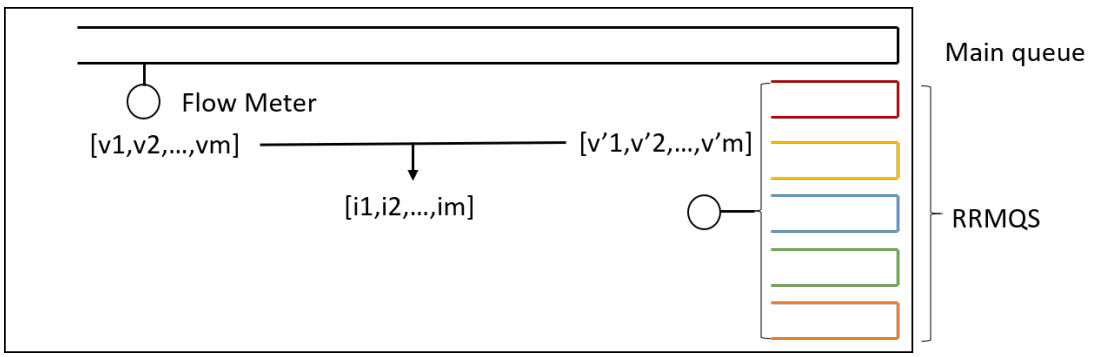

Figure 3.2: Flow meter for main queue and RRMQS

A straight forward solution is to count the average per-flow packets rate. For example in main queue, I calculate enqueue packets rate in a time window (how many packets are enqueued per second) for different TCP variants, $pktRates = [pktRate_1, pktRate_2, ..., pktRate_m]$, recall that I have m different TCP variants. And I also count the number of different flows for different TCP variants $numFlows = [numFlow_1, numFlow_2, ..., numFlow_m]$ (here I can use a bloom filter to count the number of flows, and the bloom filter should be refresh before every time window). Then average packets rate per flow $pktRatesPerFlow = pktRates/numFlows$. I do the same

Algorithm 3 Dequeue

Output: packet p;

```
1: function RRMQS_DEQUEUE(RRMQS_index)/*Round-Robin Dequeue*/
2:     p = NULL
3:     for i = 0;i < m;i++ do
4:         RRMQS_index = (RRMQS_index + i)%m
5:         p = RRMQS[RRMQS_index].dequeue()
6:         if p! = NULL then
7:             return p;
8:         end if
9:     end for
10:    return p
11: end function
12:
13: draw random variable according to probability: Pr(v == 1) = prob, Pr(v == 0) = 1    prob;
14: p = NULL;
15: if v==0 then /*First try to dequeue main queue*/
16:    p = main_queue.dequeue();
17:    if p != NULL then
18:        return p;
19:    end if
20:    p = RRMQS_DEQUEUE(RRMQS_index);
21:    if p != NULL then
22:        return p;
23:    end if
24: end if
25: if v==1 then/*First try to dequeue RRMQS*/
26:    p = RRMQS_DEQUEUE(RRMQS_index);
27:    if p != NULL then
28:        return p;
29:    end if
30:    p = main_queue.dequeue();
31:    if p != NULL then
32:        return p;
33:    end if
34: end if
35: return p;
```

for RRMQS to get another set of average per-flow packets rate, $pktRatesPerFlow^{\theta}$. Finally, I can compare $pktRatesPerFlow$ and $pktRatesPerFlow^{\theta}$ to determine whether a TCP variant uses more than its fair share.

There are still many different choices to measure in the flow meter. For example, I can measure *average flow completion time* for different TCP variants in the main queue and in RRMQS. But in this project, I only implement the packets rate flow meter, which is also the most simple and straight forward solution. For example, if I want to measure *average flow completion time*, then I need to record different flows' start time and decide whether a flow has finished in the network. And as flow completion time is usually highly skewed, which makes the "average" less effective. Thus because of limited time and the above considerations, I still use the packet's rate flow meter in the experiment.

3.4 Feedback Loop

Finally, after the measuring unfairness, I can furthermore use the result $I = [i_1, i_2, ..., i_m]$ to prioritize different flows. For example, if one TCP variant uses more than the fair share (i_l is larger than a threshold), then I just drop its packets until i_l falls. Or if one TCP variant uses less than the fair share (i_l is smaller than a threshold), then I make its packets higher priority in the main queue.

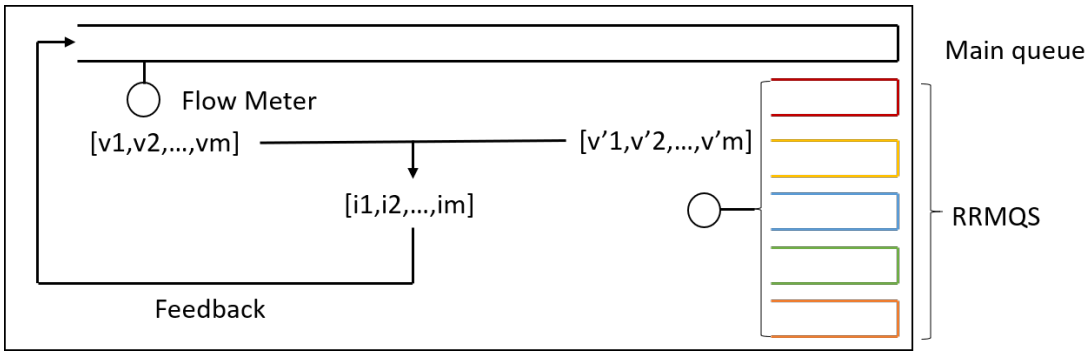

Figure 3.3: Feedback Loop

Similar to the many different metrics to measure in the flow meter, there are many different ways to feedback in the feedback loop. And the choice of reacts to what I measure will affect how well I can ensure fairness. For example, if one protocol is particularly unfair to other protocols, I can delay its packets instead of drop it. Such different solutions will have different impacts on different CCAs. For example, TCP Reno is sensitive to packets loss but not the change of delay; however TCP Vegas is sensitive to the change of delay. And thus delay the packets will have a larger impact if the protocol is Vegas instead of Reno.

Chapter 4

Evaluation

I use the "ns-3" simulator to implement the queue discipline (including RRMQS, flow meter and feedback loop), and simulate a small network with different setting of flows.

4.1 Evaluation Setting

4.1.1 Topology

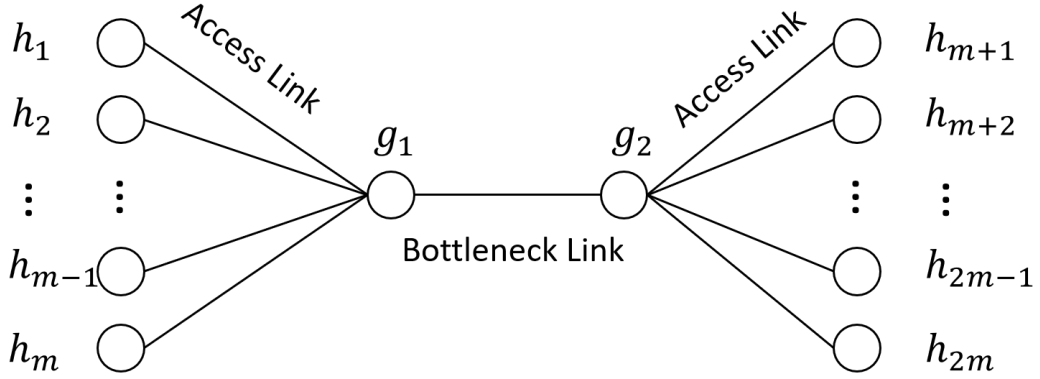

Figure 4.1: Topology

In my topology, I have $2m+2$ nodes: $h_1 \ldots h_{2m}$ and g_1, g_2. $h_1 \quad h_{2m}$ are 2m hosts, $g_1 and g_2$ act as gateways. Hosts and gateways are connected through access link with 1000 Mbps bandwidth and 20 ms delay. To observe congestion in the network, I need a bottleneck link, and g_1 and g_2 are connected through bottleneck link with 100 Mbps bandwidth and 10 ms delay.

Each host in the left side h_i, $i \quad m$, is a sender, and the corresponding receiver is h_{m+i}. Because data flows are from g_1 to g_2, I implement my queue discipline on g_1.

Each host in the left side ($h_1 \quad h_m$) uses a fixed TCP variant. For example flows from h_1 uses TCP variant 1: TCP Bic, and h_2 uses TCP variant 2: TCP NewReno... In the experiment I have $m = 12$ different TCP variants. All the 12 TCP variants is shown in Table 4.1.

h_1	h_2	h_3	h_4	h_5	h_6
TCP Bic	TCP NewReno	TCP HighSpeed	TCP Htcp	TCP Hybla	TCP Illinois
h_7	h_8	h_9	h_{10}	h_{11}	h_{12}
TCP Ledbat	TCP Scalable	TCP Vegas	TCP Veno	TCP Westwood	TCP Yeah

Table 4.1: TCP variants

4.1.2 Queue Discipline Settings

As I have $m = 12$ different TCP variants in the network, I use 12 queues in RRMQS and one main queue. I set flow capacity of each queue in RRMQS C_f to 2, which means that there will be at most 2 flows in each queue of RRMQS. I set the size of the main queue to 50 times the size of the combined RRMQS queues. Thus when dequeue, the probability to dequeue the main queue is 50/51, and the probability to dequeue RRMQS is 1/51.

I have three different concrete size setting: *(i)* small queue; *(ii)* medium queue; *(iii)* large queue.

	small queue	medium queue	large queue
main queue size (packets)	1200	12000	120000
RRMQS per queue size (packets)	2	20	200

Table 4.2: Queue Size Setting

4.1.3 Flow Settings

I randomly generate flows for each node in $h_1 \quad h_m$. The number of flows for each node in $h_1 \quad h_m$ is uniformly distributed from 100 to 200 flows. Each flow will start at a random time and has a random size (uniformly distributed from 10000 bytes to 100000 bytes).

I test with two different scenarios: *(1)* all the flows have unbounded rates, means all the flows try to send as fast as possible;*(2)* some flows have unbounded rate but some flows have bounded rate, means for some flows they cannot send faster than an upper bound, such upper bound is also randomly generated (uniformly distributed from 0.01Mbps to 1Mbps).

4.2 Evaluation Result

4.2.1 Accuracy of Fairness Measurement

To measure fairness, I use the flow meter (described in section 3.3) to calculate each TCP variants average per-flow packets rate in the main queue and in RRMQS. I set the time window length as 0.5 seconds, which means my flow meter will report packets rate every 0.5 seconds. After the experiment, I average on all the reports to get an overall average packets rate.

To evaluate the accuracy of my measurement, I also calculate the "offline" result by calculating the average per-flow throughout for each TCP variant in the main queue and in RRMQS.

Finally, I can calculate Jain's fairness index of my measurement, $J_{measure}$, and the real offline result, J_{real}. By compare the Jain's fairness index, I can calculate the accuracy of my fairness measurement. I define accuracy as $\frac{min(J_{measure},J_{real})}{max(J_{measure},J_{real})} \quad 100\%$.

All flows have unbounded rates: Table 4.3 shows the evaluation results when all flows have unbounded rates. My result shows that my method is able to measure unfairness, and the accuracy is higher than 90%.

	small queue	medium queue	large queue
$J_{measure}$	0.927	0.896	0.776
J_{real}	0.922	0.859	0.721
Accuracy	99.5%	95.9%	92.9%

Table 4.3: Jain's fairness index when all flows have unbounded rates.

Some flows have bounded rates: Table 4.4 shows the evaluation results when some flows have bounded rates. My result shows that when some flows have bounded rates, the accuracy of my measurement is lower than all flows have unbounded rates. The reason is that I pick some flows as samples in RRMQS, but the number of samples may not be enough to avoid high bias, because flows have different rates.

	small queue	medium queue	large queue
$J_{measure}$	0.953	0.944	0.779
J_{real}	0.948	0.857	0.719
Accuracy	99.5%	90.8%	92.3%

Table 4.4: Jain's fairness index when some flows have bounded rates.

4.2.2 Improved Fairness with Feedback Loop

When evaluate the accuracy of my fairness measurement, I didn't use the feedback loop. In this part, I enable my feedback loop, and to evaluate the effect of the feedback loop and how well I can ensure fairness.

I set my feedback loop to act according to the result of my flow meter $[i_1, i_2, ..., i_m]$ (recall that $I = V/V^{\emptyset}$, where V is average per-flow packets rate in the main queue, $V^{\emptyset}$ is average per-flow packets rate in RRMQS). If for a TCP variant j, if its result $i_j > 1.2$ in the last time window, I drop all j's packets in the next time window; if its result $i_j < 0.8$ in the last time window, I set j's packets higher priority in the next time window.

All flows have unbounded rates: Table 4.5 shows Jain's fairness index when all flows have unbounded rates before and after the use of the feedback loop. My result shows that the feedback loop can ensure fairness with $J > 0.99$.

	small queue	medium queue	large queue
$J_{\text{without feedback}}$	0.922	0.859	0.721
$J_{\text{with feedback}}$	0.996	0.993	0.992

Table 4.5: Jain's fairness index when all flows have unbounded rates.

Some flows have bounded rates: Table 4.6 shows Jain's fairness index when some flows have bounded rates before and after the use of the feedback loop. My results show that after using

the feedback loop, fairness is improved. However, the performance is much lower than when all flows have unbounded rates. The reason is that the performance of the feedback loop relies on the accuracy of my fairness measurement. As the accuracy of measurement is lower than when all flows have unbounded rates, the performance of the feedback loop is consequently poor.

	small queue	medium queue	large queue
$J_{\text{without feedback}}$	0.948	0.857	0.719
$J_{\text{with feedback}}$	0.975	0.878	0.881

Table 4.6: Jain's fairness index when some flows have bounded rates.

Chapter 5

Outlook

This work is a first attempt trying to let the network itself to provide service of fairness between different protocols, which used to be each protocol's task. Currently, the network do not provide service of fairness, and protocol designers need to work a lot to ensure a new protocol is fair to exist protocols. On the one hand, my "in-network" method can improve fairness and achieve better performance than the current "on-host" method. On the other hand, by letting the network take over the task of ensuring fairness, the protocol designers will worry less about the fairness problem, and new protocols will be much easier to adopt.

The design of the flow meter and feedback loop still have a lot of room for improvement. The flow meter and feedback loop I used in the evaluation part is just a very first prototype. There is a lot of different choices of metrics in the flow meter, and a lot of different ways to provide feedback, meaning the accuracy of fairness measurement and the performance of feedback loop has great potential to improve. To be honest, the performance of ensuring fairness is somewhat poor when some flows have bounded rates, but it can be improved by modifying the flow meter and feedback loop.

The method I proposed still needs to be verified that it is able to be implemented on network devices. Programmable network devices offer such possibilities, but they still have many restrictions. Thus verification of whether my algorithm is feasible in current network devices is needed.

Finally, the idea of measuring slow down caused by competition to measure fairness can be extended to more than TCP protocols. The method of the main queue and RRMQS should be applicable to a lot of protocols more than TCP. Thus, there is potential for the network to provide service of fairness for all different flows and protocols, just like a scheduler in an operating system, to supervise resource allocation and ensure fairness.

Chapter 6

Summary

I propose a fairness measurement algorithm for network devices to measure unfairness between different TCP variants, and a corresponding queue management algorithm to improve fairness between different TCP variants. My evaluation result shows that the fairness measurement algorithm has an accuracy higher than 90%, and the queue management algorithm is able to improve fairness under a complex network environment.

Bibliography

[1] Bachl, M., Fabini, J., and Zseby, T. Cocoa: Congestion control aware queuing. *arXiv preprint arXiv:1910.10604* (2019).

[2] Chiu, D.-M., and Jain, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems 17*, 1 (1989), 1–14.

[3] Fejes, F., Gombos, G., Laki, S., and Nádas, S. Who will save the internet from the congestion control revolution? In *Proceedings of the 2019 Workshop on Bu er Sizing* (2019), pp. 1–6.

[4] Mishra, A., Sun, X., Jain, A., Pande, S., Joshi, R., and Leong, B. The great internet tcp congestion control census. *Proc. ACM Meas. Anal. Comput. Syst. 3*, 3 (Dec. 2019).

[5] Turkovic, B., Kuipers, F. A., and Uhlig, S. Fifty shades of congestion control: A performance and interactions evaluation. *arXiv preprint arXiv:1903.03852* (2019).

[6] Ware, R., Mukerjee, M. K., Seshan, S., and Sherry, J. Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (2019), pp. 17–24.

[7] Ware, R., Mukerjee, M. K., Seshan, S., and Sherry, J. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of the Internet Measurement Conference* (2019), pp. 137–143.

Appendix A

My appendix

A.1 Each TCP Variant's Fairness Index

In the evaluation section, I provide Jain's fairness index $J(V, V^{\emptyset}) = \frac{(\sum_{i=1}^{m} v_i/v_i^{\emptyset})^2}{m\sum_{i=1}^{m} v_i^2/v_i^{\emptyset 2}}$. However, Jain's fairness index only gives a overall fairness metric and does not provide e.g. which one is higher and which one is lower. To show the accuracy of the measurement, I provide each TCP variant's fairness index $I = V/V^{\emptyset}$.

		i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}
$I_{measure}$	small queue	1.29	0.86	1.06	0.84	1.03	0.62	0.69	1.33	1.08	0.86	0.79	1.64
I_{real}	small queue	1.30	0.86	1.09	0.85	1.05	0.56	0.65	1.45	1.05	0.82	0.76	1.51
$I_{measure}$	medium queue	1.97	0.96	0.96	0.95	0.96	0.95	0.96	1.00	0.38	0.95	0.97	0.87
I_{real}	medium queue	2.09	1.00	0.99	0.98	1.00	0.99	0.99	1.01	0.15	0.98	1.01	0.79
$I_{measure}$	large queue	2.47	0.83	1.06	1.07	1.06	0.65	0.41	1.57	0.52	0.83	0.61	0.86
I_{real}	large queue	2.66	0.81	1.09	1.12	1.08	0.56	0.29	1.70	0.43	0.79	0.56	0.86

Table A.1: Each TCP variant's fairness Index when all flows have unbounded rates.

		i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}
$I_{measure}$	small queue	1.04	1.17	1.09	1.16	1.22	0.79	0.59	1.42	0.73	1.13	0.98	0.85
I_{real}	small queue	1.27	1.00	1.12	1.11	1.17	0.81	0.76	1.53	0.87	0.63	0.86	1.09
$I_{measure}$	medium queue	0.97	0.94	0.96	1.06	1.37	0.97	1.51	1.10	0.53	1.16	0.71	0.97
I_{real}	medium queue	0.66	1.36	0.81	1.57	2.04	1.21	1.54	0.47	0.43	1.49	1.12	1.07
$I_{measure}$	large queue	1.27	1.24	1.08	0.87	1.47	1.18	0.93	3.11	0.46	0.88	0.88	0.86
I_{real}	large queue	0.73	1.24	1.73	0.75	1.47	2.17	0.72	3.15	0.33	0.73	0.99	0.68

Table A.2: Each TCP variant's fairness Index when some flows have bounded rates.