Lecture 19: QoS Part I: Queueing Delay for Single Flow, Policing flows via Leaky Buckets

This lecture and the next one will go into much more detail than your book. Please read the lecture notes.

1 Administrative

1. Project 3 – You should be finishing up. Umair (umair@cs.cmu.edu) has been extremely generous with providing unlimited office hours for people with questions. Please go see him!
2. HW 3 – being handed back today.
3. What's left in course: Project 4 and Homework 4.
4. Project Partners – Occasionally people want to switch partners, or else their partners may have dropped the class and they need a new partner. If you need a new partner please see me after class.
5. Please see me to pick up old homeworks or midterm if you haven't yet.

2 Motivation: Real-time applications

Real-time applications are those for which **timeliness** of data is very important. A typical example is a phone conversation where participants must be able to **respond** and be heard almost immediately.

Question: It's hard to carry a conversation if the time between when you speak and a listener hears you exceeds .3 second. Have you ever noticed this problem when calling a far-away country?

Question: Which of these would you classify as real-time applications: *telnet*, *ftp*, *robot control*, *email*, *Web browsing*, *watching a video*, *interactive video*, *interactive games*, *IP telephony*?

Answer: *Ftp*, *email* and *Web browsing* are not generally considered real-time. *telnet* could be considered to be real-time, since a telnet session is not usable if delay exceeds a few seconds. *Robot control* is real-time because the robot must be able to respond to commands instantly to keep it from crashing. Likewise *interactive games* are real-time. *IP telephony* is a real-time application in that the total delay from speaker to listener has to be less than .3 seconds, or else the sentences become unintelligible. *Watching a video* is not considered real-time (you can rely on an initial 5-minute buffer), however any *interactive video* is certainly real-time.

Where is delay created?

Question: What causes delay (what are the components of delay)? What causes variability in delay?

Answer: There's propagation delay that can't be helped. There's also transmission delay. These two types of delay are constant. Most importantly there's queueing delay. This delay is variable.

Recall the function of a router. Packets from various flows arrive at a router. They are partitioned based on the output link that they will leave on. There are many flows leaving on each output link. Each output link has associated with it an **output queue**. The rate at which packets leave the output queue is constrained by the bandwidth on that link. Today's Internet uses primarily **FCFS** (first-come-first-served) scheduling at the output queue. This means that packets are put on the wire in the order that they arrive into the output queue. The output queue is constrained in length, and therefore is typically referred to as a **router buffer**. An incoming packet will be dropped if there is no more room in the buffer.

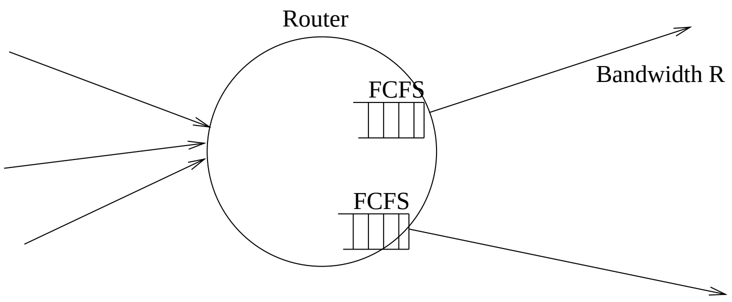

Figure 1: *There's an output queue for each outgoing link.*

Question: Given FCFS queueing at the routers, what kind of worst-case delay and loss guarantees can we hope to make?

Answer: No guarantees. Observe that any flow may experience anywhere from 0% packet loss to 100%, depending on how many **competing flows** show up at the same time. Assuming that a packet is not dropped, its delay within the router could be quite long – as long as the time for the entire output queue to drain. TCP recovers the losses by retransmitting, but TCP can not make delay guarantees.

High-level goals:

For real-time applications it is desirable to have an absolute **delay guarantee** – e.g. the time to get from the source to the destination will never exceed 100 ms. Observe that we seek a **worst-case** bound, not an average-case analysis. Sometimes it is also desirable to have a worst-case bound on the amount of **loss** (some loss is usually not a problem). Related to the problem of achieving absolute bounds on the delay is the problem of achieving absolute bounds on **delay jitter** – the maximum difference in delays experienced by any two consecutive packets.

The problem of how to achieve delay guarantees is *not solved*. Several potential solutions have been proposed and we will discuss these, and their pros and cons.

The problem of delay is a complicated one. In order to really understand what's happening, we have to start with the question of what happens when you have a **single flow** [1]. We will study a single flow in detail. We will find that even for a single flow there can be significant **queueing delay**. Worse yet, this single flow can have a significant effect on the delay of other flows. We will discuss mechanisms for **policing** a single flow to control its variability, which in turn will allow us to control its effect on other flows. We will then move to the problem of controlling delay in **multiple flows**. Today's lecture will be more theoretical. Wednesday's lecture will describe some specific protocols that have been proposed to provide applications with guaranteed delay.

2.1 Rate Definitions

Before we begin, a few definitions are in order. When we talk about the **mean arrival rate** into a queue, we will be talking about how many packets/sec arrive at the queue, e.g. $\lambda = 30$ packets/sec.

Question: The mean **interarrival time** is defined how?

Answer: $\frac{1}{\lambda} = \frac{1}{30}$ seconds between packets.

The quantity $\frac{1}{\lambda}$ is a mean. There may be variability in the interarrival times. We say that the interarrival times are **constant** or **deterministic** if there is no variability in the interarrival times. I.e., the time between arriving packets is exactly $\frac{1}{30}$ seconds.

[1] A *flow* is a set of packets all having the same source process and having the same destination process and following the same path.

3 Queueing delay in a single flow

Suppose that a flow generates packets at an average rate of $\lambda = 3$ packets per second. The flow passes through a single router with bandwidth $\mu = 6$ packets per second where $\lambda < \mu$. There are no other flows.

Question: What is the **queueing delay** experienced by the packets of our flow?

Answer: It depends on the variability in the arrival stream of the flow:

Let's first consider the case where the flow generates packets with evenly-spaced interarrival times, a.k.a. constant interarrival times (time between packets is exactly $\frac{1}{\lambda}$). The time to drain a packet will be $\frac{1}{\mu}$, where

$$\frac{1}{\lambda} > \frac{1}{\mu}$$

Thus there will never be any packets in the flow's queue. Zero queueing delay.

Now suppose the flow's packets don't have constant interarrival times. Instead they arrive according to a **Poisson process with rate** λ. The average interarrival time is still $\frac{1}{\lambda}$ seconds between packets (the average arrival rate is still λ), but there is variability in the arrival stream. Now we can see that we could have queueing delay, if a burst of packets arrives at once. How bad can the average queueing delay to be now?

To answer this, we need to understand what's a Poisson process with rate λ. Also need to understand a little Queueing Theory. We're therefore going to digress for a half hour and give a tutorial on queueing theory, for those who haven't ever seen it (rest can sleep). At the end of the tutorial we'll come back to this question of the queueing delay.

4 Brief Queueing Theory Tutorial

We spend this section providing a very brief and oversimplified introduction to the field of Queueing Theory. Queueing theory is a beautiful area of mathematics involving stochastic analysis which allows us to prove theorems about systems performance, such as predicting queue sizes, delays, throughput, etc.., given the system workload. Queueing theory is generally used for modeling whole networks or complex systems. In this introduction we will only look at a single-server or router.

4.1 The Server Model

A server represents a resource, e.g. bandwidth.

Consider the following simple example. Packets arrive to a router at mean rate $\lambda = \frac{1}{6} = \frac{1 packet}{6 seconds}$. The bandwidth rate on the outgoing link of the router is $\mu = \frac{1}{3} = \frac{1 packet}{3 seconds}$. This example would be modeled by the single-server queueing network shown in Figure 2. In the queueing network in Figure 2, λ is referred to as the **mean arrival rate into the server**, and μ is referred to as the **mean service rate of the server.** A "job" here corresponds to a packet and "servicing the job" corresponds to putting the packet on the wire. The time to "service a job" is the transmission time.

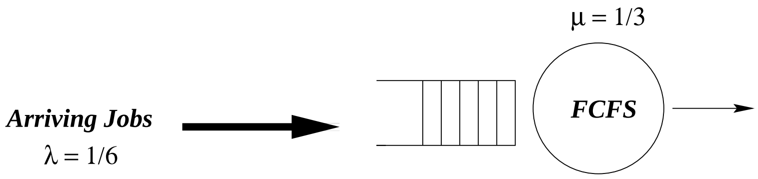

Figure 2: Single-server Network.

Throughout our discussion we will assume that the **service time** has a *constant* distribution with mean $\frac{1}{\mu}$. That is, the service rate is μ and it takes exactly $\frac{1}{\mu}$ seconds to service a packet (put a packet on the wire).

This is representative of bandwidth. The **arrival rate** could be a **constant rate** λ. This would mean that the spacing between packets is exactly $\frac{1}{\lambda}$. Alternatively the arrival rate might be a Poisson Process with rate λ, or some other process.

4.2 What's a Poisson Process?

A Poisson process is a random process which looks very much like coinflips. The simplest definition of a Poisson process with rate λ is to imagine time discretized into tiny timesteps of size dt. At every dt timestep, we flip a coin, which has probability λdt of coming up heads. Arrivals occur at the points where we have heads.

Question: By this definition, what's the mean time between arrivals?

Answer: Mean number of flips between heads is $\frac{1}{\lambda dt}$. Mean time between flips is dt. Therefore, mean time between arrivals is

$$\frac{1}{\lambda dt} \cdot dt = \frac{1}{\lambda}$$

as desired.

What's important to observe about a Poisson process is that the interarrival time can range from 0 to ∞ (certainly not a constant interarrival time). Real network flows are often much burstier even than a Poisson process, but a Poisson process is a good model to start with.

4.3 Queue lengths

Question: In the server in Figure 2, what happens if $\lambda > \mu$?

Answer: If $\lambda > \mu$ the queue size goes to infinity over time.

Question: If the interarrival times and the service times had a **constant**, a.k.a. **deterministic**, distribution, what would the queue length be at the server?

Answer: 0.

Question: If the arrival process is a Poisson process with rate λ and the service time is constant $\frac{1}{\mu}$, what's the average queue length at the server?

Answer: It turns out that this answer depends on the quantity $\frac{\lambda}{\mu}$.

Question: What do you think $\frac{\lambda}{\mu}$ represents?

Answer: The **load** at the server, or the mean server **utilization**. Typically this is denoted by $\rho = \frac{\lambda}{\mu}$.

When you take a queueing class you will prove that if the arrival process is a Poisson process with rate λ and the service time is constant equal to $\frac{1}{\mu}$, then the average queue length is

$$E[\text{Number packets in queue}] = \frac{\rho^2}{2(1-\rho)}$$

Throughout the rest of this lecture, we will concentrate on the case where the arrival process is a Poisson process and the service time is constant.

Question: What does the E[Number of packets in queue] look like as a function of load ρ?

Answer: When load is low, mean queue length is low. So long as load stays below around 0.5, the mean queue length is close to zero. However as load increases beyond that point, mean queue length grows rapidly, accelerating quickly, as shown in Figure 3.

Note how mean queue length skyrockets when λ is close to μ. This translates to the case where desired bandwidth is close to (but still less than) available bandwidth. It's the **variability** in the arrival process that is causing the high queue sizes. In the Internet, variability in the arrival times is even more bursty than for a Poisson process, and thus the queue length created by a single flow can be even worse than what we've shown.

Momentarily, we'll consider the relationship between queue length and delay, but first we need to make sure that everyone understand the term **throughput**.

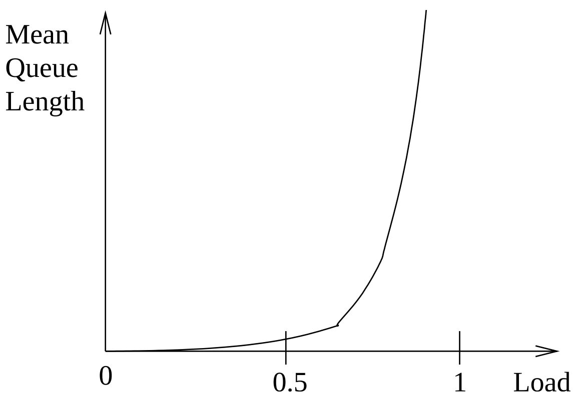

Figure 3: Mean queue length as a function of load.

4.4 Understanding Throughput

Question: What is throughput?

Answer: Throughput is the mean rate at which packets complete service.

Question: Consider the two single-server systems shown in Figure 4. Which system has higher **throughput**? Which system has higher queueing time?

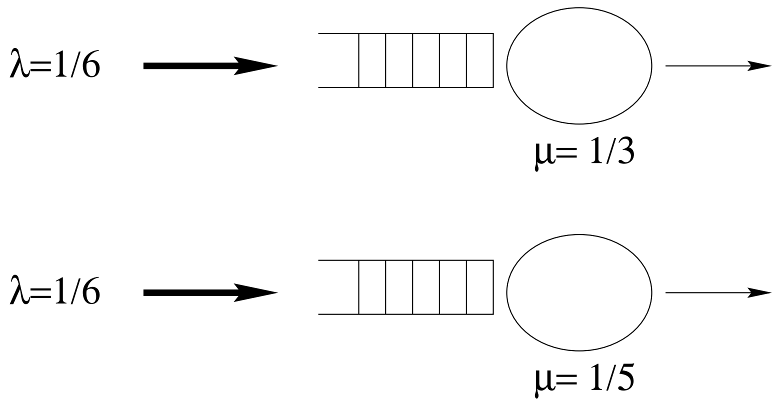

Figure 4: Comparing throughput of two systems.

Answer: The *throughput* of the two systems is the same: $\frac{1}{6}$. Although the top system has a higher service rate (bandwidth), the number of packets per second that pass through is still limited by the arrival rate. The *mean delay* at the first server is much lower however, as the load is lower, and the mean queue length is lower.

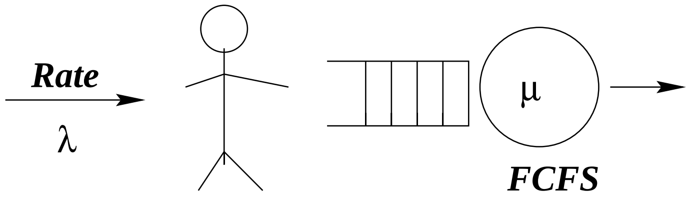

Figure 5: Little's theorem applied to a single server.

4.5 Mean delay and Little's Law

The delay a packet experiences is the time it spends in the queue. Obviously **mean delay** is is in some way proportional to **mean queue length**.

Question: Can you guess what the exact relationship is?

Answer: Little's Law states that the average number of jobs queued in any system is equal to the product of the average arrival rate into the system and the average time a job spends waiting in queue in the system.

Theorem 1 *For any ergodic system we have that:*

$$E[N_Q] = \lambda E[T_Q]$$

where $E[N_Q]$ is the expected number of jobs in queue in the system; λ is the average arrival rate into the system; $E[T_Q]$ is the mean time a job spends waiting in queue in the system.

It is important to notice that Little's theorem makes no assumptions about the arrival process, the service time distributions at the servers, the network topology, the service order, nothing!

Proving Little's Law requires a whole lecture. Instead, let us try to come up with some intuitions to help understand Little's Law.

Question: Any ideas for why Little's Law holds?

Here's the way I always remember Little's theorem (Intuition only): Think about a single FCFS queue, as shown in Figure 5.

- Customer arrives, sees $E[N_Q]$ in the queue.
- Expected time for each customer to complete is $1/\lambda$ (not $1/\mu$), since the average rate of completions is λ (see throughput discussion above).

- So expected time until customer leaves is $E[T_Q] = \frac{1}{\lambda} \cdot E[N_Q]$.

Applying Little's Law we see that, if the arrival process is a Poisson process with rate λ and the service time is constant rate μ, then the average queuing delay is:

$$\begin{aligned} E[Delay] &= E[\text{Number in queue}] \cdot \frac{1}{\lambda} \\ &= \frac{\rho^2}{2(1-\rho)} \cdot \frac{1}{\lambda} \end{aligned}$$

Again, the $1-\rho$ term in the denominator tells us that this gets quite high when λ is close to μ. If the arrival stream is burstier than a Poisson Process, this is even worse.

5 Policing flows — Leaky Buckets

From the previous section, we can see that a single flow (no other flows in the system) can already experience a high queueing delay, even though the bandwidth available to it is more than sufficient ($\lambda < \mu$). This is due to the variability in the arrival process in a flow.

A flow is created by an application. The stream of packets stemming from this application may be bursty (variable interarrival times). If this is the case than we cannot satisfy our goal of bounding the delay for this flow. **Worse yet, once we bring in other flows, we cannot bound the effect that our flow will have on other flows** (the queue length due to our flow alone can get very high).

In order to limit the effect that our flow has on other flows, we need to remove the variability in our flow. To do this we use a **leaky bucket**.

A **leaky bucket** or **token bucket filter** is a mechanism for controlling both a flow's mean rate and its maximum burst size. The **max burst size** refers to the maximum number of packets that can be sent into the network over an extremely short period of time. A leaky bucket is applied at the flow source. Once the flow passes through the leaky bucket, it is completely predictable. Figure 6 shows a leaky bucket policing mechanism. The bucket can hold up to b tokens. New tokens are added to the bucket at a constant rate of r tokens per second. Before a packet is transmitted into the network, it must first remove a token from the leaky bucket. If the bucket is empty, the packet is blocked, waiting on a token.

Question: For the leaky bucket shown in Figure 6, what is the maximum burst size?

Answer: b

Question: For the leaky bucket shown in Figure 6, what is the maximum number of packet that can enter the network in *any* interval of length t?

Answer: $rt + b$

This is in fact the **definition of a leaky bucket with parameters (r,b)**, namely a policing device that ensures that during any period of time t the maximum number of packets that can enter the network in *any* interval of length t is $rt + b$.

Question: What is the average rate at which packets enter the network?

Answer: $lim_{t\to\infty} \frac{rt+b}{t} = r$.

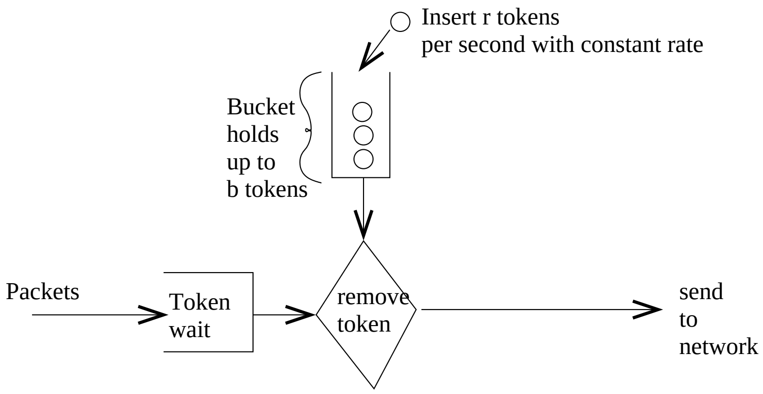

Figure 6: *Leaky bucket with rate r and bucket size b at flow source.*

The **average rate** limits the amount of traffic that can be sent into the network over a relatively long period of time. It is also possible to define a **peak rate** constraint. The peak rate constrains the maximum number of packets that can be sent over a shorter period of time. This is a further constraint on the allowable burst.

Question: Suppose we want to police a flow to an average rate of 6000 packets per minute, while limiting the peak rate to 1500 packets per second. At the same time we want to limit the (instantaneous) burst size to 3000 packets. Is this possible?

Answer: A solution is shown in Figure 7. Observe that the maximum number of packets which leave in t minutes is $6000t + 3000$ as desired for *any* t. There may be two seconds at peak rate.

6 Providing delay bounds for multiple flows

Question: Once flow i has been policed with leaky bucket parameters (r_i, b_i), what re-

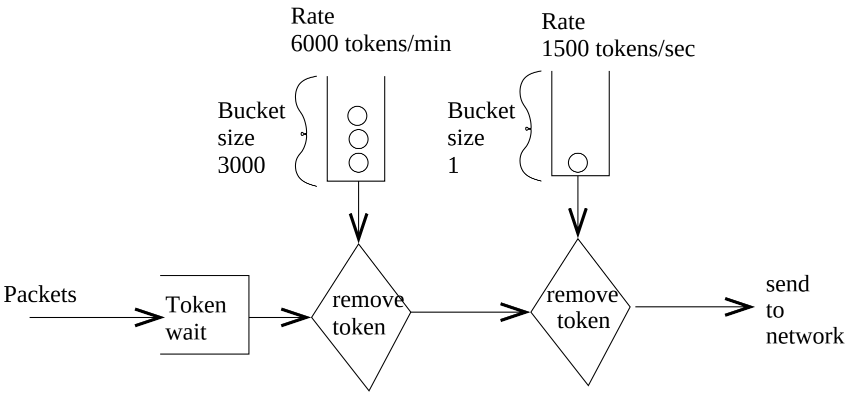

Figure 7: *Policing the mean rate, the peak rate, and the max burst size on a flow.*

sources does flow i need along each router in its path?

Answer: r_i packets/sec worth of outgoing bandwidth at each router, and b_i buffer space at the first router after the policer. Note, no buffer space is needed at subsequent routers assuming that the bandwidth flow i receives is doled out at a constant rate (constant time to service a packet). In practice, it is not always possible to split up the bandwidth among flows so that the bandwidth available to flow i is exactly constant-rate, as we'll discuss soon, and so some buffering is necessary at later routers as well.

What's nice about the above, is that once flow i has been policed, if it can make the reservations it needs, then the delay a packet of flow i experiences after it leaves the policer is totally predictable.

Each flow i now goes about trying to make the necessary reservation in terms of bandwidth and buffer space at each router. This is called the **call admission** process where a flow declares its QoS requirements and then is either admitted to the network or blocked. We'll explain the details of this proposed protocol next time.

If flow i is admitted, from flow i's perspective, the network looks like Figure 8. **The receiver of flow i is guaranteed to receive an average rate of r_i packets per second, with a maximum burst size of b_i.** Observe that the delay at each router is simply $\frac{b_i}{r_i}$ where r_i is specified in packets per second.

Question: How do the multiple flows get served at the output queue of a router?

Answer: To handle multiple flows, all of which want to leave through the same output queue, we need to have **isolation between flows** so that each flow is able to maintain its guarantees. Therefore we need a **separate** buffer for each flow. The output buffer for flow i is a buffer of size b_i. See Figure 9.

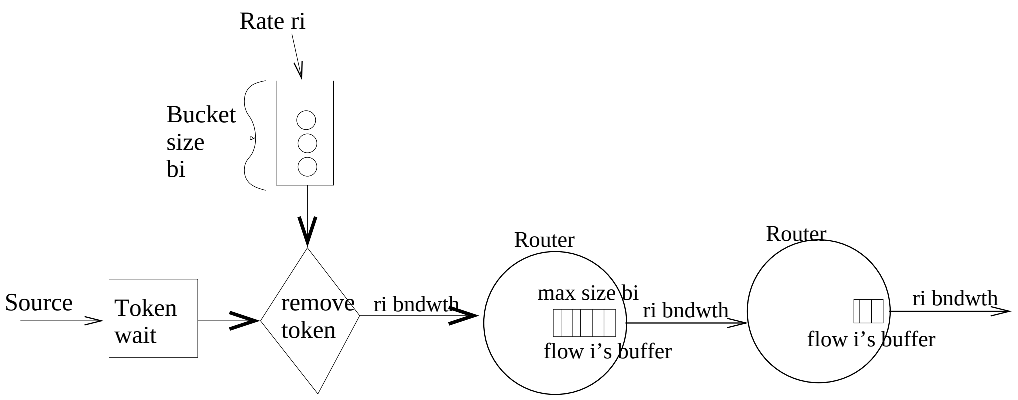

Figure 8: *Flow i's perspective of the network.*

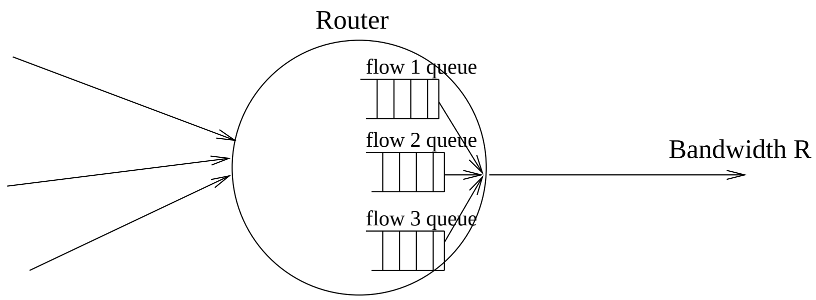

Figure 9: *Separate output queue for each flow.*

The only question that remains is how to split up the available bandwidth R into the required bandwidths: r_1, r_2, $\ldots$, r_n, so that each of the flows receives a constant rate of bandwidth, and so that any excess bandwidth is proportioned fairly among the flows.

In general this is not always possible. However there is a pretty good approximation called **Weighted Fair Queueing (WFQ)**, which we will discuss next time.

Thank you!