Lecture 20: QoS Part II: WFQ, Intserv and RSVP, Diffserv

Note: This is the second part of a 2-part lecture. The lecture notes for QoS cover more material than is available in your book. You are responsible for the material in the notes.

Last lecture we were working on the QoS problem (quality of service): How to provide real-time applications with the delay guarantees that they need?

We started out looking at a single flow. We saw that it could have unlimited queueing and delay due to the variability in its arrival process. Worse yet, when the flow competes with other flows for the same outgoing link, its unlimited queueing will cause unlimited delay for the other flows as well. We then saw we could police a flow so that its mean rate was r and its max burst size was b. Once a flow is policed, with parameters (r, b), it only needs r share of bandwidth at each router and at most b buffer space at each router. We can now ensure that:

1. No packets of the flow will be lost.
2. The delay of the flow can be bounded (the max delay at each router is simply the reserved buffer length divided by reserved bandwidth in pkts/sec.). To get total delay need to add on propagation delay to this.
3. The effect of our flow on other flows is limited to our flow's buffer requirement and its bandwidth requirement at each router. Our flow can no longer have an unbounded effect on other flows.
4. The receiver of our flow is able to provision for our flow, since our flow follows simple specifications.

Figure 1 shows 3 flows sharing the output link of a router. Note that the bandwidth of the router and the buffer space of the router is being shared.

One assumption that we have made so far is that we can easily **split** the bandwidth R on a link into the desired reservations: $r_1, r_2, r_3, \ldots$ where $\sum r_i \leq R$ so that each flow i is served at **constant rate** r_i. This sounds easy enough but it's actually not!

Weighted Fair Queueing (WFQ) is an algorithm which allows us to get *close* to constant rate splitting. We describe it below.

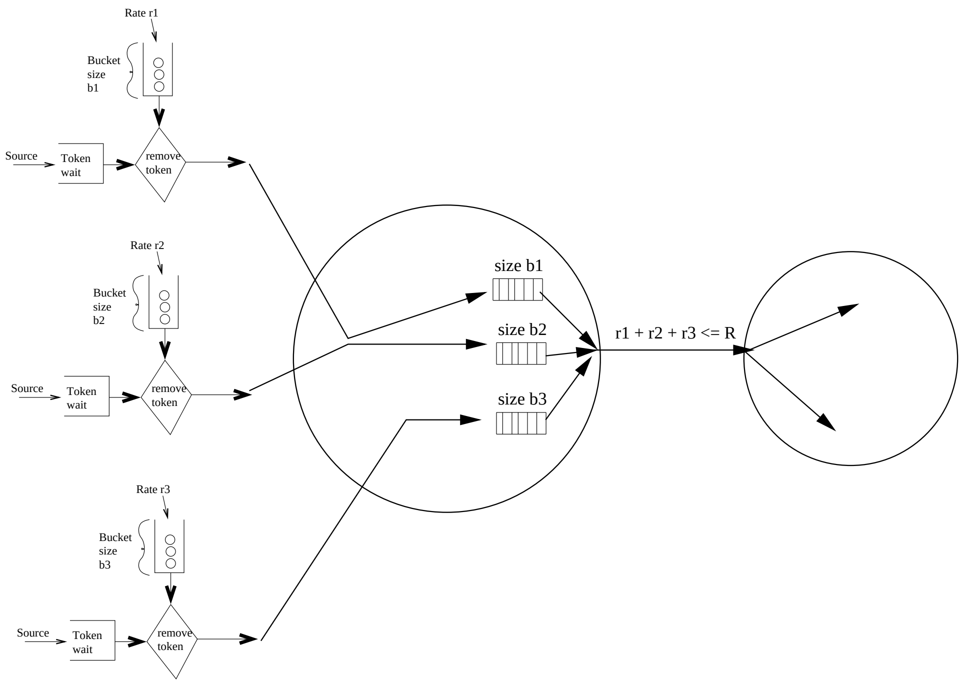

Figure 1: *Three flows sharing the output link of a router.*

1 WFQ via GPS

Consider the following motivating example: Suppose there are 3 flows: A, B, and C. A needs $\frac{1}{2}$ the total bandwidth. B needs $\frac{1}{3}$ the total bandwidth. C needs $\frac{1}{6}$ the total bandwidth. The bandwidth is disbursed on a **per-packet** basis. We can imagine a "wheel" specifying who's turn it is to send a packet.

Question: Is it possible that all the flows receive their bandwidth at a constant rate?

Answer: No. See Figure 2 below. No matter how we lay out the turns on the wheel, observe that A and B can't both receive service at a constant rate.

This example raises several concerns:

1. It is not always possible to give each flow a constant rate bandwidth.
2. It's not obvious how in general the "turns" are assigned. Imagine that you have many flows and the fraction of bandwidth they want are: $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{12}$, $\frac{1}{24}$, and $\frac{1}{24}$. We need to assign the flows turns to best approximate constant rate bandwidth for each. What's an algorithm for doing this? How close to constant rate does it get you?

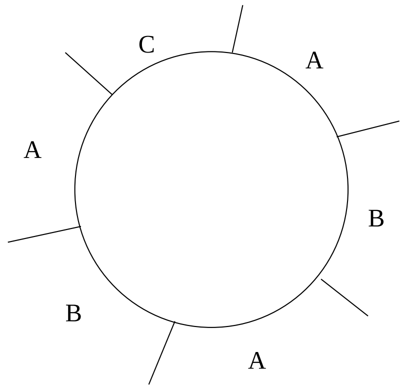

Figure 2: *Handing out bandwidth.*

3. To make things more complicated, not all packets are the same size. How are the turns assigned when packets have different sizes, for example B's packets are half the size of A's packets.

4. Lastly, in general the total reserved bandwidth may be less than the available bandwidth. The remaining bandwidth needs to be distributed fairly among the flows.

The solution for all these problems is called WFQ. WFQ is modeled after a hypothetical algorithm, **Generalized Processor Sharing (GPS)**, which doesn't really exist, but is a fluid abstraction. WFQ is often referred to as a **packetized approximation** to GPS. (Hence WFQ is also known as PGPS – Packetized GPS).

We'll work through two examples, so you can see how this works. In the first example, Figure 3, we assume that all packets have the same size.

In the fluid (GPS) world, we imagine that packets are infinitesimally small and all flows drain through a pipe. During the time it takes for flow A to drain 1 unit, flow B will drain 2/3 of a unit and flow C will drain 1/3 of a unit. In WFQ, the next packet picked is always that packet which would complete service first under GPS, assuming that no new packets arrive.

Since the first packet of A would be the first to complete in GPS, it is the first packet picked by WFQ. At this point, we can imagine that flow B has 2/3 of a packet completed and flow C has 1/3 of a packet completed. Flow B would be the next to complete a packet under GPS. During the time that flow B completes its packet, flow A would complete another 1/2 a packet, and flow C would complete another 1/6 of a packet. The next flow to complete a packet would be flow A. During that time, flow B would complete 1/3 of a packet and flow C would complete another 1/6 of a packet. The next 3 packets to complete would be

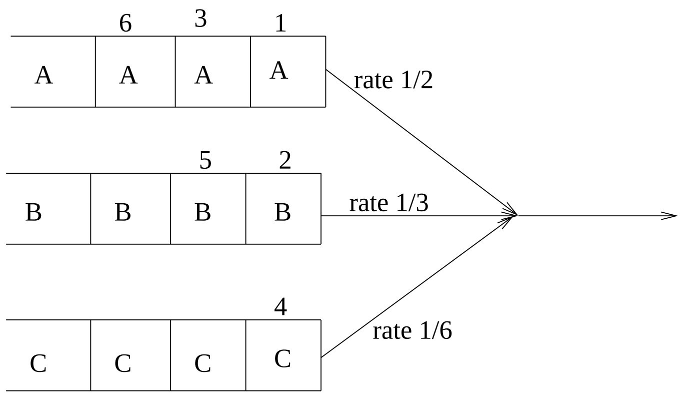

Figure 3: *Packetized version of GPS. Numbers show order in which packets get to drain.*

from flow C, flow B, and flow A. These all complete simultaneously, so we pick an arbitrary order (C, B, A). The process continues.

Observe that WFQ does not produce constant rate service – in fact that's impossible for this example. It is however an algorithm for how to take turns and get close to constant-rate service.

Now, let's suppose that packets have **different sizes**.

Question: How does the algorithm change?

Answer: The algorithm doesn't change. The answer will of course change.

Example: Two flows A and B arrive at a router with a WFQ scheduling policy. The WFQ scheduling is modeled after GPS. Flow A has reserved $\frac{3}{4}$ of the bandwidth on the outgoing link. Flow B has reserved $\frac{1}{4}$ of the bandwidth on the outgoing link. Flow A's packets are twice the size of flow B's packets. Classify each of the first 5 packets to leave the router as being from flow A or flow B. Show your work.

Answer: A B A B A The first packet comes from flow A. At this point there is a packet from flow B with 2/3 of the work done. The second packet comes from flow B. At this point there is a half-done packet from flow A. The third packet comes from flow A. At this point there is a packet in flow B which is 1/3 done. The fourth and fifth packets come from Flow B and A (in any order).

It turns out that one can prove that if WFQ is done according to the above algorithm, then the maximum delay a packet experiences is not far from what we would have if every flow received bandwidth at a constant rate. In a grad class I would ask you to prove this. This was proven by Parekh and Gallager. There have been hundreds (probably thousands) of papers written on WFQ and its variations. I am handing out a survey paper by Hui Zhang (one of our faculty) which describes some of these.

WFQ provides each flow its bandwidth reservation, with close to constant rate. Let's return one more time to the experience of flow i, policed by a leaky bucket with parameters (r_i, b_i). The leaky bucket assures that flow i is transmitted with average rate r_i and maximum burst size b_i. Observe that at the first router there will be a maximum burst size of b_i, where b_i is the bucket size of the leaky bucket policer. The first router therefore needs to allocate a buffer of size b_i for flow i (to ensure zero loss). The first router also has to reserve bandwidth r_i for flow i. If WFQ was actually able to provide **constant rate bandwidth** r_i at all the routers, then we wouldn't actually need a buffer for flow i at subsequent routers after the first router, and there also wouldn't be any queueing delay at any of after the first router. However, since (i) the bandwidth provided may be $> r_i$ and (ii) WFQ is not perfect, we actually do need some buffer at each of the subsequent routers. In general the queueing delay for flow i at a router with buffer size b_i (for flow i) and constant-rate bandwidth reservation r_i is $\frac{b_i}{r_i}$. If the bandwidth reservation r_i doesn't have constant rate, a little more queueing delay is experienced, and buffers need to be increased a little. If you want to understand the details see RFC 2212 and the Parekh Gallager 93 paper.

2 Intserv and the RSVP protocol

Intserv (short for Integrated Services) is a framework developed within the IETF to provide individualized quality of service guarantees to **individual** application sessions (flows). Above we have described the algorithmic components necessary to create these guarantees. We now describe the mechanism or **protocol** by which a flow goes about making the reservations that it needs at the routers. The protocol used is called **RSVP** which stands for Resource ReSerVation Protocol. RSVP is described in RFC 2205 which is over 100 pages. I will just explain the highlights here.

RSVP allows flows to reserve **resources** which means link bandwidth and router buffers. We will mostly concentrate on the bandwidth reservations.

RSVP allows for a more general framework of flows than what we have up to now assumed in this class. I'm going to have to digress for a few minutes and explain this more general framework, called **multicast**.

So far in this class we've looked only at the problem of delivering packets from a single source to a single destination. This is called **unicast**. Many network applications require delivery of packets from a sender to a **group of receivers**. For example, the video, audio or text of a live lecture may be being streamed to many participants. **Multicast** is defined as the sending of packets from one sender to multiple receivers.

Question: Suppose we have one sender and 3 receivers, as shown in Figure 4(a). How could we implement multicast without making any change to the existing routers?

Answer: We could just send 3 unicast flows.

Question: Why would sending unicast flows waste bandwidth?

Answer: It could happen that multiple copies of the same packet will be sent over the same link (because the same link is used in getting from the sender to all 3 receivers.

Question: So what would we like to have happen?

Answer: We want to send only one copy of a packet for as long as possible.

This is in fact exactly what happens. The sender sends the message to a **multicast address** which is a class D **group address** which represents all the receivers. Special **multicast-enabled routers** are used to route packets from the sender to the class D address. These routers are "smart" – they duplicate packets only when necessary, as shown in Figure 4(b).

We will have a whole lecture on multicast next week. For now, just assume that routers are all multicast-enabled. (Note: unicast is just a degenerate case of multicast).

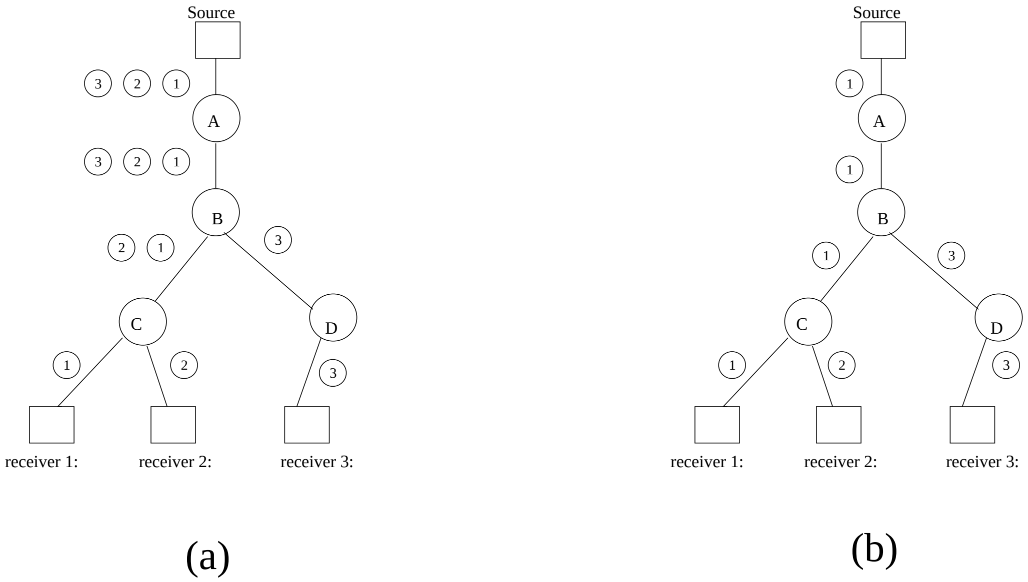

Figure 4: *(a) Multicast implemented as 3 unicast flows. (b) Multicast implemented efficiently using multicast routing. Note that many fewer packets are sent in (b) – less bandwidth used.*

We're now ready to describe RSVP. The high-level view is as follows: A flow creates an **Rspec** which defines the specific **bandwidth**, r, and **buffer size**, b, that it requires. The flow uses the RSVP protocol to communicate its RSpec to all routers in its path. These routers in turn need to make a decision if they can give the flow what it wants (that process is called **call admission**). If so, the network will make the appropriate **reservations** and admit the flow. If not, the network will ask the flow to come up with a more modest request.

To implement RSVP, RSVP software must be present at the sender(s), receiver(s), and routers. Two distinguishing characteristics of RSVP are:

1. It can reserve bandwidth/buffers in **multicast trees**, not just for single flows.
2. It is **receiver-oriented** in that the receiver of a data flow will initiate the resource reservation.

This is best explained via an example. Suppose we have a multicast flow as shown in Figure 5. A sender might be sending a video or audio stream to a large group of people. The sender sends the video to the multicast group address identifying these people. The video is being sent along a multicast tree. The receivers are the leaves of the tree in Figure 5.

Question: Why does the reservation need to be initiated by the *receivers* rather than by the source?

Answer: It's the receiver who determines at which rate she can receive the video (based on her technology). She also knows what the variability is that she can tolerate (max burstiness – jitter) and the amount of loss that she can tolerate. Suppose a receiver can receive at 128 Kbps. Then she will try to make reservations for the flow to send her video at rate 128 Kbps. Either the routers will have this bandwidth available (and the buffer space), in which case the flow will now come to the receiver at 128 Kbps. Alternatively, one or more of the routers on the path may not have 128 Kbps of bandwidth available. In this case the receiver might have to settle for receiving the video at only 64 Kbps because that's as fast as the network can send it. It is therefore the receiver who specifies the desired rate of the flow.

Question: Suppose that receiver 1 in Figure 5 can receive flows at 128 Kbps and receiver 2 can receive at 28 Kbps. What is the desired reservation for the flow (assume the flow is a video)?

Hint1: If we only reserve 28 Kbps for the flow, then the 128 Kbps receiver has to suffer with low quality video.

Hint2: If we reserve 128 Kbps for the flow, how does the 28 Kbps receiver watch the video?

Answer: The solution is to make the reservation for the **maximum rate of all the receivers**, in this case 128 Kbps. The 28 Kbps receiver is still able to watch the video because the video is encoded in **layers**, as you saw in your multimedia lecture. For example,

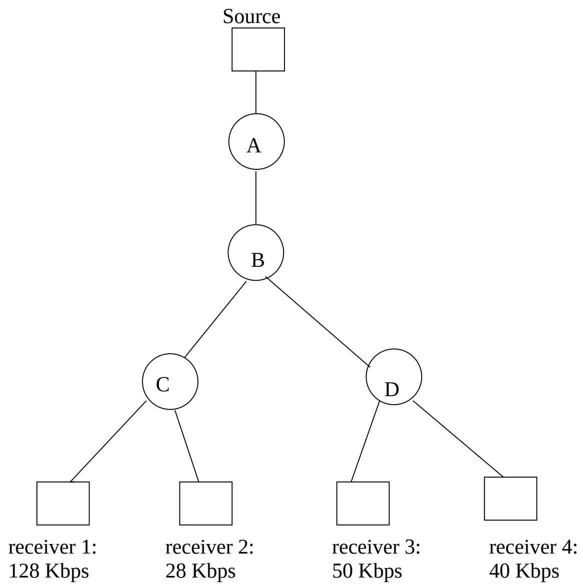

Figure 5: *Multicast tree on which we run RSVP.*

the video might be encoded in 2 layers, a base layer (of rate 20Kbps) and an enhancement layer (of rate 100Kbps). Thus receivers with 28 Kbps access could receive the low-quality base-layer image, and receivers with the 128 Kbps could receiver both layers to construct a high-quality image.

The receivers must communicate to the source their receiving rates. Each receiver sends a **reservation message (Rspec)** upstream into the multicast tree, specifying the rate at which the receiver would like to receive data from the source. Router C in Figure 5 will receive reservation requests of 28 Kbps and 128 Kbps. It will try to reserve 128 Kbps (the max of these) on the link between routers B and C. Similarly, router D will receive reservation requests of 50 Kbps and 40 Kbps and will try to reserve 50 Kbps on the link between routers B and D. Router B will receive requests of 50 Kbps and 128 Kbps. It will therefore try to reserve 128 Kpbs on the link between A and B.

Observe that the sender does not need to know the receiving rates of all the receivers. It only needs to know the **maximum** rate of all its receivers. The sender then encodes the video in multiple layers, and sends all layers into the multicast tree.

We have been describing reservation messages as just being reservations for bandwidth, but the receiver's reservation message can also specify **jitter** requirements. This will eventually translate into requiring certain leaky bucket policing parameters on the source and buffer reservations throughout.

If at any point the reservations cannot be made, a message goes back to the *receivers* who have to adjust their demands and try again. This message is called a **ResvError message**.

Question: Suppose there's one receiver request that's bigger than all the rest (in our example, 128 Kbps). Until this receiver lowers its requirements, none of the small reservations can get through. It doesn't seem fair to punish all the small reservations when they would be serviceable if it weren't for this one big request. What can be done?

Answer: ResvError messages can be set up to establish a state in routers known as the **blockade state**. The blockade state in a router modifies the merging procedure to omit the offending reservation from the merge, allowing the small requests to be forwarded and established.

So far we have assumed that the receiver knows the particular upstream path along the *multicast tree* to the sender.

Question: How does the receiver know that path along the multicast tree to the sender?

Hint: The source knows how to reach all the receivers because it can send a message to the multicast group address and all the routers know how to get it there.

Answer: An **RSVP path message** is sent from the sender to all the receivers. The purpose of the path message is to "leave breadcrumbs" showing the receiver how to get to the sender along the tree. As each router receives the path message, it records the address of the previous router, and thereby it knows the **router just upstream from it**. In this way the receivers are able to send their merged reservation requests all the way up to the sender.

Some logistics: The RSVP messages are sent hop by hop from router to router directly over IP. The RSVP message is placed in the payload field of the IP datagram. The messages have to stop at each router for the router to process the message. A message is processed by the RSVP protocol which has **protocol number 46**.

Question: Explain the above. What exactly goes in the headers?

Answer: A message starts at a host (the receiver). The message is encapsulated with an IP header which has the host's IP as the source address and the first router's IP (next hop upstream) as the destination address. The protocol field in the IP header is set to 46. When the message gets to the first router it is unwrapped and sent up to protocol 46 which either makes a reservation or merges reservations or does whatever RSVP is supposed to do. After that this first router encapsulates the packet in an IP header again. This time the source address is that of the first router. The destination address is that of the second router, and again the protocol field in the IP header is set to 46, etc.

A few more comments before we leave RSVP:

Throughout this discussion, we have assumed that the receivers require some bandwidth r and some maximum variability parameter b. It could also be the case that the receiver requires some maximum total **end-to-end delay** d for the flow. In this case routers must also keep track of **propagation delay** when deciding whether to accept the flow (call admission). This does not appear to be discussed in any of the RFCs.

Throughout this discussion, we have assumed a single sender. In general it is of course possible to have **multiple senders**, each sending to (possibly overlapping) multicast groups simultaneously. For example, I might be a receiver who is watching a video (sent by one sender) while listening to an audio tape sent by another sender, and playing an interactive game, sent by a third sender. I am part of all three senders' multicast groups.

There are several types of bandwidth reservations a receiver can make, given that she is receiving data from multiple senders. We distinguish between:

1. Fixed-filter style reservation
2. Shared-explicit style reservation.

The **fixed-filter style reservation** is exactly what we would imagine bandwidth reservation to be like with multiple senders: The receiver specifies a list of senders from which she wishes to receive data along with a bandwidth reservation for each sender. These reservations are **distinct**. For example, I may be a receiver with bandwidth 120 Kbps and I may choose to split this over 3 reservations each of 40 Kbps.

In the **shared-explicit style reservation**, the receiver again specifies a list of senders from which she wants to receive data, along with a **single** bandwidth reservation, to be shared among all the senders in the list.

Question: When would you use a shared reservation?

Answer: You might be listening to several senders who are unlikely to transmit simultaneously. For example, suppose you're having a conference and transmitting all the audio spoken by each person. There could be many speakers, involved in the session, but since everyone is hearing everyone else, a limited number of people will talk at once. Each receiver might issue a shared-explicit reservation for 2 times the bandwidth required for a single sender (to allow for overspeaking).

Question: When would you use the fixed-filter reservation?

Answer: Suppose you're having a video conference between people in different locations. Video showing each of the 5 people you're talking to is up on your screen. You need to reserve separate bandwidth for each of these video streams so that they are clear.

3 Difficulties of Intserv

In case you haven't noticed, implementing Intserv is quite complicated.

Question: What are some issues with respect to implementation?

Answer: The problem is one of **scalability**. All routers must be RSVP capable. Each router maintain **per-flow state** for each flow passing through the router. This could be as many as 256,000 flows per minute.

Sadly, after 7 years of work on Intserv and RSVP and thousands of papers, most people have given up on making Intserv happen in practice. Instead people are turning to a much simpler, but far less powerful model, called **Diffserv**.

4 Diffserv

The **Diffserv** (Differentiated Services) working group within the IETF is trying to come up with a simpler form of delay guarantees. Their goal is to only have simple functionality at the routers and let the complex operation happen at the edge of the network. Diffserv is described in RFC 2475.

The idea is that flows may only be of one of a small number of types: e.g., Platinum, Gold, Silver, or Bronze. A flow negotiates for a type with its ISP (by paying $$). This negotiation is done at the edge router. Each packet of the flow carries the type in the TOS header field under IPv4 (under IPv6, this would be the "Traffic class field"). Every router looks at the type and schedules the flow based on its type.

One scheduling policy is for the router to perform some sort of **priority queueing**, where the router always gives highest priority to Platinum packets, second priority to Gold packets, etc.. An alternative scheduling policy is for the router to use **reservations** and allocate a certain fraction of the bandwidth on its outgoing link to each class, with the highest fraction of bandwidth allocated to Platinum and the lowest fraction of the bandwidth allocated to Bronze.

Question: In what way is the router state reduced under Diffserv as compared with Intserv?

Answer: While under Intserv, a router has to maintain state for each flow, under Diffserv, router needs only to keep track of 4 priority classes. All packets within the same priority class are **indistinguishable**, regardless of their flow.

Question: What kind of delay guarantees does Diffserv provide – consider both types of scheduling policies above.

Answer: It doesn't. In both the case of the priority queueing policy and in the case of the fixed bandwidth reservations packets have to compete with flows in their own priority class, and there are no restrictions on how much traffic those flows can create.

Question: In fact under one of these scheduling policies it's actually possible that Platinum flows will have much worse performance that say Gold flows. How is that possible?

Answer: Suppose that we use **reservations scheduling** and that Platinum flows are allocated 50% of the link's bandwidth and Gold flows are allocated 25% of the link's bandwidth. However suppose that the traffic intensity of the Platinum flows if 20 times higher than that of the Gold flows. In this case, the Gold flows will actually have better average performance than the Platinum flows, since they receive a higher fraction of the bandwidth.

Question: How can Diffserv provide any guarantee at all then to Platinum flows which have paid so much money?

Answer: One idea is to limit the number of Platinum flows, so that they don't interfere with each other. A related idea is to make each flow agree to a fixed **traffic profile** whereby there's a limit on its average rate and possibly peak rate (remember last lecture's example with 2 leaky buckets), so that there's no single Platinum flow hog, which ruins performance for all the other Platinum flows. Still, it's not clear how end-routers can provision for what the **interior routers** will see, since you can't predict how many flows will simultaneously need to access the same interior router.

Another **difference** between Diffserv and Intserv is that under Intserv the **receivers** make the reservations whereas under Diffserv the **sender** for the flow purchases its packet type. Under Diffserv, I can request that all packets I send are Platinum class, which has priority over all other classes. However I can't say anything about the delays of packets I receive, since those may be coming from varied senders.

This is an extremely active area of research. People are determined to find a way to charge customers more for better service. It's still open however, exactly how to guarantee that "better service" in a simple fashion without much state. I'm on the Sigmetrics program committee and we got 10 submissions this year all proposing new ideas for making Diffserv or Intserv work. One of the main difficulties is one of logistics: to make Diffserv work, we need the ISP's to cooperate in terms of the definition of gold, silver, and bronze. For example, if I pay my ISP a lot of money to get gold service, then I want to know that I will receive gold treatment when I'm being routed through all parts of the Internet, including through other ISPs. You can see how the other ISPs may not want to agree to this ...

Question: So it sounds like Diffserv doesn't provide delay guarantees and also can be difficult to coordinate in terms of getting ISPs to agree to the Gold/Silver/Bronze system. What are people proposing be done?

Answer: The direction of current research is to bring the server closer to the client. That is, companies like Akamai replicate the server (or cache) in many locations, so no matter who the client is, a server (or cache) will be close to you. This is one way to give you more predictable delays. Also, if the server is in your ISP, you may even be able to have different classes of service. Srini will give the lecture on Content Distribution.

I'm handing out two research papers on the topic of Intserv and Diffserv by a professor on the CMU faculty, Hui Zhang.

Thank You!