1

1. PIE reduces queue sizes by dropping packets early, so short web request flows incur less queueing delay.
2. 10 flow case, because there is more statistical multiplexing.

2

1. Congestion windows will be roughly the same, because CWND = throughput * rtt and TCP's RTT unfairness ratio.
2. $S_1 \to h_1$ has the larger drop rate. Use the TCP throughput equation, with Flow A = $h_5 \to h_1$, Flow B = $h_2 \to h_1$, Flow C = $h_4 \to h_3$.

   $p_{ss} = \frac{1}{T_C^2 RTT_C^2}$

   $p_{sh} = \frac{1}{T_B^2 RTT_B^2}$

   We know, $T_C = 100 - x$ and $T_B = 100 - x$ since A and C share a bottleneck link with capacity 100, and the same for A and B. So, $T_C = T_B$. Since $RTT_C > RTT_B$, we can conclude that p_{sh} is greater.

3

Let the loss probability along $S2 \to S1$ be p_1 and $S1 \to h1$ be p_2. Then flow $h4 \to h3$ has loss p_1, $h2 \to h1$ has loss p_2 and $h5 \to h1$ has loss $1 - (1 - p_1)(1 - p_2)$ (1− the probability that a packet is *not* dropped by *both* links). Since p_1, p_2 are small, $p_1 p_2 \approx 0$, so the $h5 \to h1$ loss is $\approx p_1 + p_2$.

The RTTs for each flow, including queueing delay, are: $R_{43} = 120, R_{51} = 140, R_{21} = 60$. Combined with the TCP throughput equation, we get a total of 5 equations:

$$T_{51} = \frac{C}{R_{51}\sqrt{p_1 + p_2}} \qquad T_{43} = \frac{C}{R_{43}\sqrt{p_1}} \qquad T_{21} = \frac{C}{R_{21}\sqrt{p_2}}$$

$$T_{51} + T_{43} = 100 \qquad T_{51} + T_{21} = 100$$

C is a constant of proportionality, but we won't need it. The units here won't wind up being terribly important, but to be more precise, you can think of throughput in units of Kb/ms, which is identical to $Mbps$.

The last two equations imply that $T_{43} = T_{21}$. Solving for p_1 and p_2 in the first two equations and simplifying, we get:

$$\frac{1}{49T_{51}^2} = \frac{1}{9T_{21}^2} + \frac{1}{36T_{21}^2} \qquad T_{21} = \frac{7\sqrt{5}}{6} T_{51}$$

Plug this into the last two equations to get:

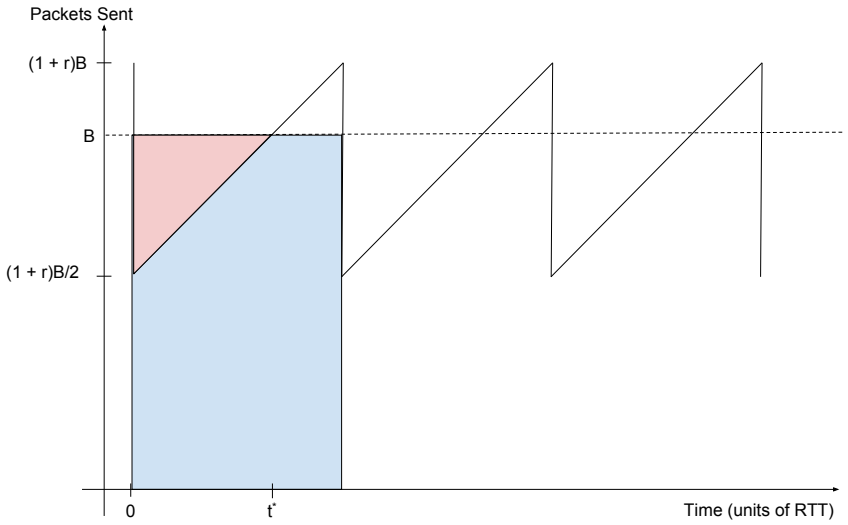
$$T_{51} = \frac{600}{6 + 7\sqrt{5}} \approx 27.7 \text{ Mbps} \qquad T_{43} = T_{21} = \frac{700\sqrt{5}}{6 + 7\sqrt{5}} \approx 72.3 \text{ Mbps}$$

4

1. If buffer is 1 BDP, then the window right before a loss is 2 BDPs, so after the loss event, it's cut down to 1 BDP, which still achieves full utilization in the steady state. For this router, that's 40 Gbps $\cdot$ 0.1 s = 4 *Gb*.

2. Think of a sawtooth diagram. If the peak of the sawtooth occurs at 1 BDP, the minimum is at 0.5 BDP. Over the course of 1 sawtooth cycle, the flow sends an average of 0.75 BDP.

3. Consider a sawtooth whose maximum is $(1 + r)$ BDPs, shown below.



Until time t^*, the flow is underutilizing the link. Consider the *deficit*, the number of extra packets it could have sent until the link was fully utilized. That deficit is the area of the red shaded triangle. If B is the BDP in packets, the size of this triangle is $\frac{1}{2}\left(\frac{1-r}{2}\right)^2 B^2$ packets. Since there are $\frac{1+r}{2}$ RTTs in one full period of the sawtooth, the total number of packets needed to fully utilize the link (blue + red area) is $\left(\frac{1+r}{2}\right) B^2$. Therefore, the utilization is:

$$U(r) = \frac{\left(\frac{1+r}{2}\right) B^2 - \frac{1}{2}\left(\frac{1-r}{2}\right)^2 B^2}{\left(\frac{1+r}{2}\right) B^2} = \frac{-r^2 + 6r + 3}{4 + 4r}$$

A quick sanity check confirms that $U(0) = \frac{3}{4}$ and $U(1) = 1$.

Note: we've made the simplifying assumption that all RTTs are equal over the course of one sawtooth. This isn't technically true, since queueing delay will increase the RTT after t^*, but it simplifies the analysis.

5

The new XCP update procedure is of the form $f = p_i - n_i$, where f is the advised change in window size, and:

$$p_i \propto \frac{\Delta}{\sum_j \frac{1}{\lambda_j}} \cdot \frac{r_i}{\lambda_i} \qquad n_i \propto \frac{\Delta}{\sum_j s_j} \cdot s_i r_i$$

where r_i, λ_i are the RTT and throughput of flow i, respectively, and Δ is the total throughput change determined by the efficiency controller. The effect on the efficiency controller depends on assumptions you make about the traffic.

Under the assumption that there are many flows, one flow changing either r_i and λ_i will have minimal impact on the average statistics computed at the switch and will therefore not have an effect on the efficiency controller. We'll only focus on the fairness implications below.

(a) Forging RTTs has no effect.

If the sender forges its RTT and reports r_f instead of the true value r, then its $p_i \propto \frac{r_f}{\lambda}$ will be proportionally larger (or smaller). However, the sender corrects for the misreported RTT when updating its *cwnd*, which cancels out the effect of an altered p_i. This forgery also has no effect on other flows, which only depend on the proportionality constant $\frac{\Delta}{\sum_j \frac{1}{\lambda_j}}$ and have no dependency on the reported RTTs. An analogous argument holds of the negative feedback: though n_i will change if RTTs are forged, the sender compensates to produce the expected throughput.

(b) Forging throughput only affects fairness on additive increase.

If the sender forges its throughput and reports λ_f instead of the true value λ, it may either underreport or overreport. If $\lambda_f < \lambda$ and the network is trying to increase aggregate throughput, then the positive feedback proportionality constant decreases (which sends smaller positive feedback to other flows), while p_i increases (which sends bigger positive feedback to the forging flow), so the misbehaving flow will take a larger share of the throughput during the next control interval, hurting fairness. If $\lambda_f > \lambda$, then the opposite occurs: p_i for the forging flow decreases while the feedback to other flows increases, giving the misbehaving flow a *smaller* share of the throughout, and also hurting fairness. Notice that λ doesn't figure at all in the negative feedback, so forging throughput only affects the additive increase portion of XCP.