In-network Resource Allocation

(Scribed by Ambuj Ojha)

Key Insight

Adding intelligence in routers can improve congestion control relative to purely end to end schemes as seen in TCP Tahoe and BBR. We discuss two examples of how this has been achieved.

XCP (eXplicit Control Protocol)

https://cs.nyu.edu/~anirudh/CSCI-GA.2620-001/papers/xcp.pdf

XCP is an alternative protocol which performs better than TCP on high BDP networks.

High BDP Network examples:

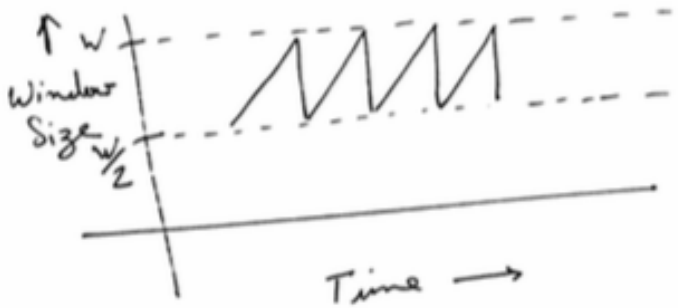
- cellular networks (Bandwidth: Few Mbps, RTT: ~100 milliseconds)
- satellite links (Bandwidth: few Kbps, RTT: ~ few seconds)
- transatlantic links (Bandwidth: few Gbps, RTT: ~few seconds)

Non high BDP Network example:
Data Center network (Bandwidth: few Gbps, RTT:~microseconds)

XCP Motivation:

Its been seen through both theory and experiments that as the product of bandwidth and latency increases, TCP becomes inefficient and prone to instability. In the below notes on next page, an empirical discussion is provided that also establishes the same.

TCP Behavior in high BDP Networks



Steady State Throughput of TCP = $\frac{\text{Avg. Window Size}}{RTT} = \frac{3W}{4\,RTT}$

Since T.C.P. Tahoe follows additive increase window size goes from $W/2$ to W in $W/2$ units of time (RTTs)

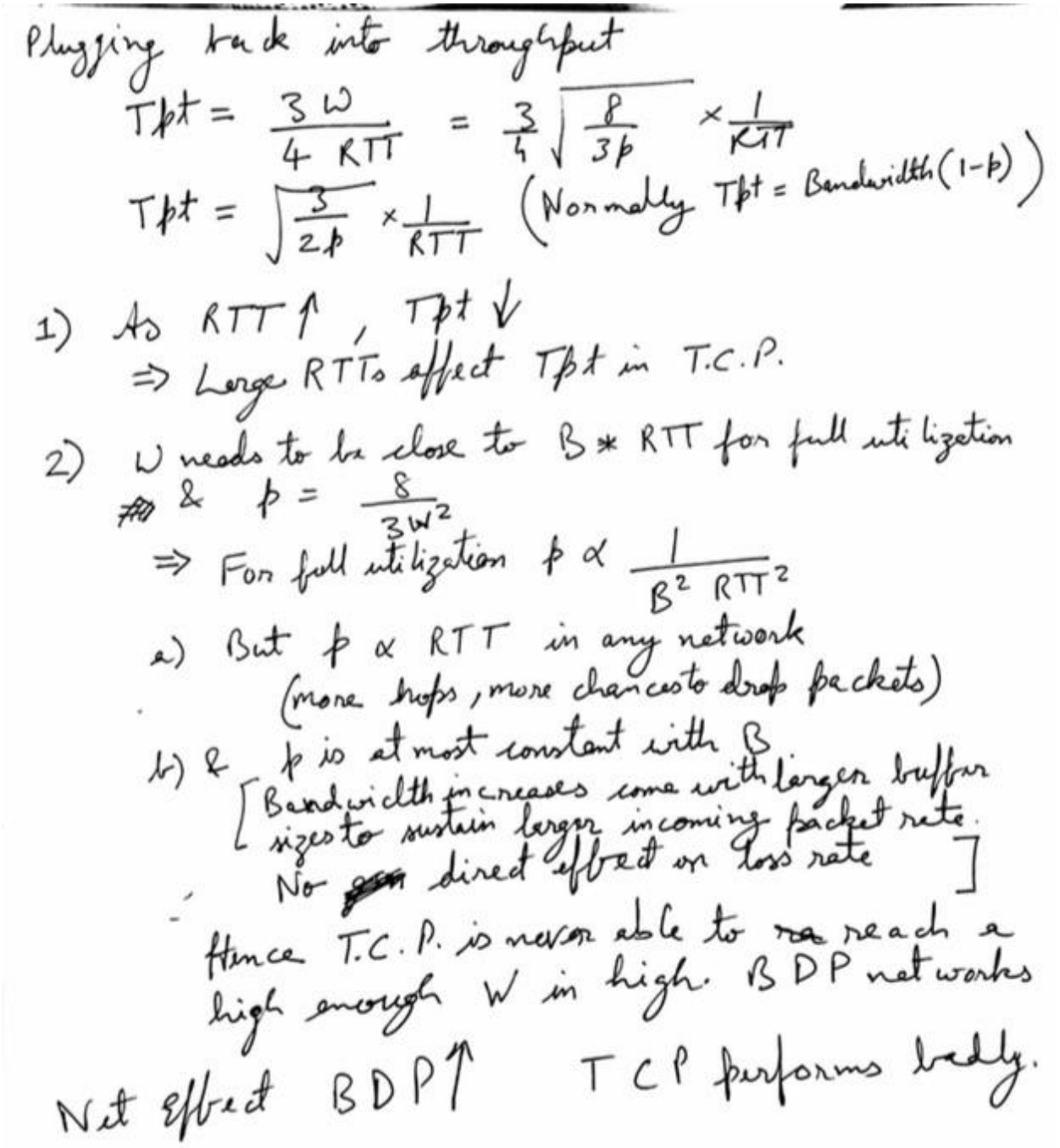
$\Rightarrow$	1st RTT	$W/2$ packets
	2nd RTT	$W/2+1$ "
	⋮	
	$W/2$ RTT	$W/2 + W/2$ "

Total Number of Packets Sent b/t 0 to $W/2$ RTTs $\approx \frac{W}{2} * \frac{W}{2} + \frac{W}{2} * \frac{W}{4} \approx \frac{3W^2}{8}$

Since 1 packet was lost,

Loss probability $p = \frac{8}{3W^2}$

$\Rightarrow W = \frac{8}{\sqrt{3p}}$



XCP Features:

- XCP Generalizes ECN (DECBit) which set a single bit in the packet header which is set explicitly to provide network congestion feedback. XCP provides much richer feedback than ECN. Some papers tried to find the minimum size of packet header needed to provide as good feedback as XCP

- Richer feedback at routers & computation at routers => Better congestion control than standard TCP
- XCP decouples efficiency and fairness (high utilization and fair distribution) with different controllers for each.
    - Utilization Controller keeps link busy
    - Fairness Controller enforces different fairness policies independent of efficiency
    - Fairness controller also divides feedback to be sent to end nodes, across multiple packets
- XCP doesn't maintain per-flow state on the router. Instead, XCP has the packets themselves carry per-flow state like RTT. This was first introduced in CSFQ in 1998.

Protocol:

- XCP provides a joint design of end-systems and routers
- Senders maintain congestion window 'cwnd' and 'rtt' and communicate these to the routers via a congestion header in every packet.
- Routers monitor the input traffic rate to each of their output queues. Based on the difference between the link bandwidth and its input traffic rate, the router tells the flows sharing that link to increase or decrease their congestion windows by annotating the congestion header of data packets. Feedback is divided between flows based on their cwnd and rtt values so that the system converges to fairness. Ultimately, the packet will contain the feedback from the bottleneck along the path.
- When the feedback reaches the receiver, it is returned to the sender in an acknowledgment packet, and the sender updates its cwnd accordingly.

Congestion Header:

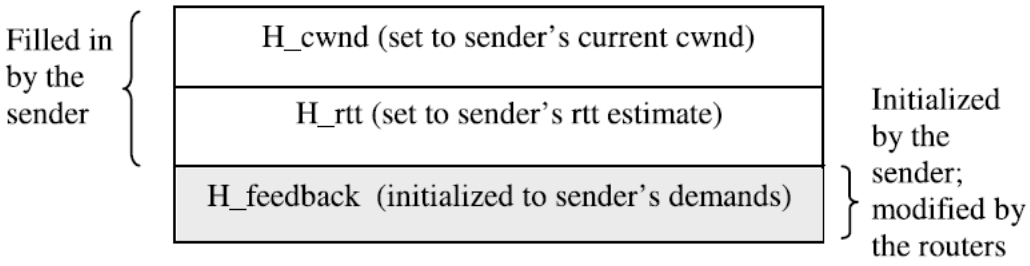


Figure 1: Congestion header.

XCP Sender:

- XCP sender maintains a congestion window of the outstanding packets, cwnd, and an estimate of the round trip time rtt.
- Whenever a new acknowledgment arrives, positive feedback increases the senders cwnd and negative feedback reduces it

XCP Receiver:

- When acknowledging a packet, it copies the congestion header from the data packet to its acknowledgment.

XCP Router:

- Computes the feedback to cause the system to converge to optimal efficiency and min-max fairness.
- An XCP router uses an *efficiency controller* and a *fairness controller* to compute feedback. Both of these compute estimates over the average RTT of the flows traversing the link, which smooths the burstiness of a window-based control protocol. The average RTT is computed using the information in the congestion header.

Efficiency Controller:

The main purpose of Efficiency Controller is to maximize link utilization while minimizing drop rate and persistent queues by taking into account only the aggregate traffic and not worry about fairness issues. It computes 'Phi' the desired increase or decrease in the number of bytes that the aggregate traffic transmits (in a control interval) based on two terms: spare bandwidth and queue size along with gain constants alpha and beta. The EC uses a Multiplicative-Increase Multiplicative-Decrease law (MIMD), which increases the traffic rate proportionally to the spare bandwidth in the system (instead of increasing by one packet/RTT/flow as TCP does). This allows XCP to quickly acquire the positive spare bandwidth even over high capacity links. All the EC requires is that the total traffic changes by 'Phi' over this control interval.

Fairness controller:

The job of the fairness controller (FC) is to apportion the feedback to individual packets to achieve fairness. It uses AIMD (additive-increase/multiplicative-decrease) to converge to fairness, but other policies are also possible. Based on the aggregate feedback ,'Phi', provided by the Efficiency Controller to the Fairness Controller-

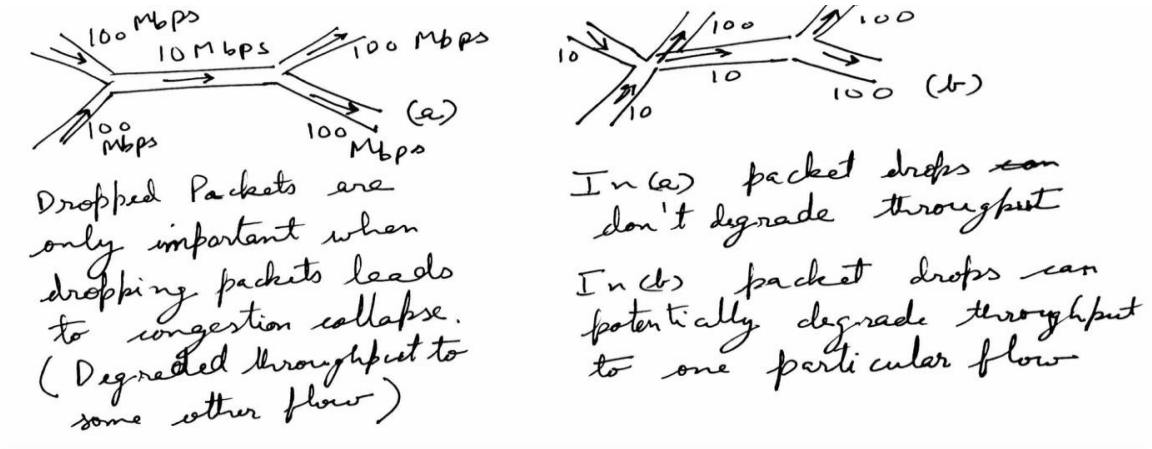
- If positive, increase throughput for all flows by a constant.
- If negative, decrease throughput for all flows proportional to their throughput.

The precise control laws follows from these two goals, after accounting for the fact that the feedback is split across packets.

Specifically XCP outperforms TCP over the following metrics:

a) Bandwidth utilization
   - XCP's Efficiency controller with MIMD helps it provide better bandwidth utilization as compared to TCP
   - Also increases in bandwidth does not affect XCP as it affects TCP

b) Reducing packet drops
   - XCP's router feedback allows the system to discover congestion before actual packet drops happen and hence XCP has much lower packet drop rates than TCP implying lower potentially wasted network capacity
   - Below diagram explains when packet drop impacts throughput



c) Reducing Queue Size
- Much lower queue sizes than TCP => less collateral damage on latency-sensitive traffic.
- Large queue size can cause collateral damage to small TCP flows (e.g. fetching a thumbnail for a website)

d) Security
- XCP has policing agents at edges of the network which detect misbehaving XCP senders that try to grab more than fair share. The agents maintain per-flow state and monitor the behavior of the flows to detect network attacks and isolate unresponsive sources.
- XCP facilitates the job of these policing agents because of its explicit feedback.
- However, this solution isn't really effective because it require sender to implement XCP

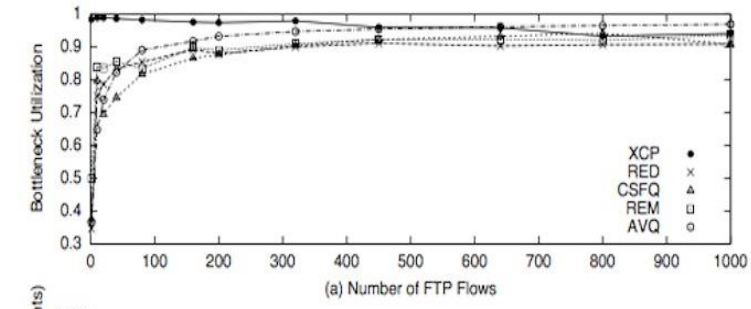
e) Convergence
- XCP dampens oscillations and converges smoothly to high utilization small queues and fair bandwidth allocation
- Whenever a new flow starts, the fairness controller reallocates bandwidth to maintain min-max fairness.
- Decoupling utilization and fairness control ensures that this reallocation is achieved without disturbing the utilization. Instantaneous queues might build up at routers, which effectively absorb the new traffic and drain quickly afterwards.
- One of the earliest papers to state rapid convergence as a design goal for congestion control
- Important because many flows (e.g.Web flows) are short-lived

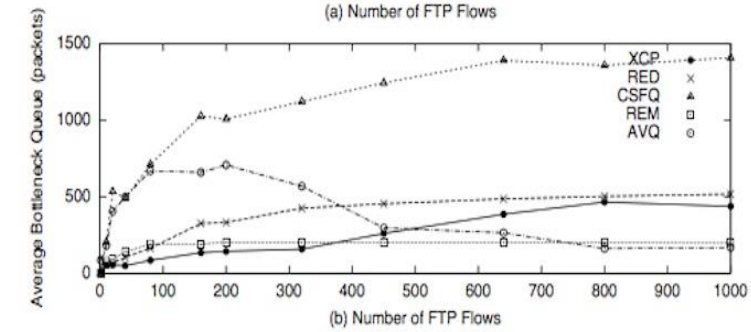
f) Stability
- Stability is defined as the ability to reach a final good state regardless of where you start.
- XCP has faster convergence to fair share after network conditions change hence greater stability.
- The analysis for stability is derived from control theory. In general, congestion control systems are treated as a form of control systems. Some parameter is observed (ex- loss, delay, rtt, rates, etc.) and is used to control some other parameter (ex- window sizes, rates, etc.) XCP was one of the first papers to make this connection precise.

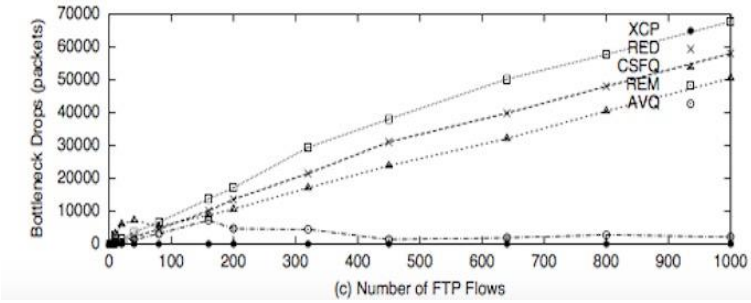
g) Scalability

Figures on the right show that as the number of FTP flows increase XCP scales well in terms of bottleneck utilization (high), queue size (low) and packet drops (low).

XCP is being implemented with a variety of queuing disciplines- RED, CSFQ, REM and AVQ

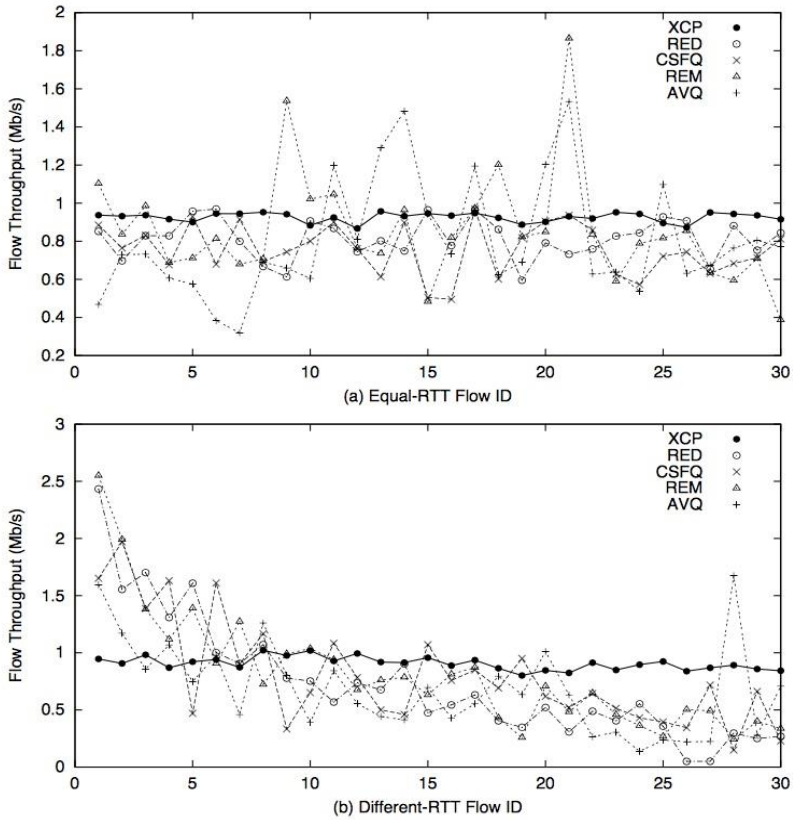






h) Fairness

- Fairness Controller in XCP implements fairness among flows
- No RTT unfairness => The intercontinental bulk transfer doesn't get squeezed out by the local flow
- Ability to use other fairness policies, such as pricing policies
- Diagram below shows XCP does not penalize high RTT flows as TCP does



Reasons why XCP wasn't adapted:

1) Every end host and router needed change without partial deployment.
2) XCP came up with a way to perform partial deployment but it's not always clear where the bottlenecks are, hence one couldn't turn on XCP in just those regions
3) It's also not clear if we are going to get partial benefits from partial deployment
4) Router implementation was tougher to ensure than end host implementation. Routing was traditionally done in hardware for speed. XCP logic needed to be added to the ASIC otherwise it would have been too slow.

However with ASICS becoming programmable XCP might make a comeback.

WFQ (Weighted Fair Queuing)

https://cs.nyu.edu/~anirudh/CSCI-GA.2620-001/papers/wfq.pdf

Historical Context:

WFQ comes last in the progression of adding more and more intelligence to routers. TCP Tahoe assumes routers do nothing except plain packet switching & routers eventually drop packets once their buffers are full. XCP makes routers compute & send feedback and run fairness and efficiency controllers, but still assumes FIFO stateless routers. WFQ adds intelligence to routers to make them decide the order in which packets are scheduled to enforce fairness on a packet-to-packet basis.

WFQ does this by saving per flow state in the router and using this state to implement a sophisticated version of round robin with better network congestion handling characteristics than TCP Tahoe or XCP.

Per Flow State:

- Finishing Time of the last packet for each flow
- Queues Data Structures for each flow

Max Min Fairness Criterion:

An allocation is fair if

1) no user receives more than its request
2) no other allocation scheme satisfying condition 1 has a higher minimum allocation
3) condition 2 remains recursively true as we remove the minimal user and reduce the total resource accordingly

WFQ Characteristics:

- Per flow state and more intelligence in the routers means we can expect more out of our network.
- WFQ can provide protection from arbitrary misbehaving sources, not just misbehaving TCP or XCP sources.
- Fairness is provided on a packet-by-packet basis under the criterion of Max-Min Fairness. XCP/TCP provide it after convergence.
- WFQ also provides isolation. If a flow sticks to its allocated bandwidth, it will have low delays and not be affected by anyone else.

Figure 1 in WFQ paper illustrates isolation very well.

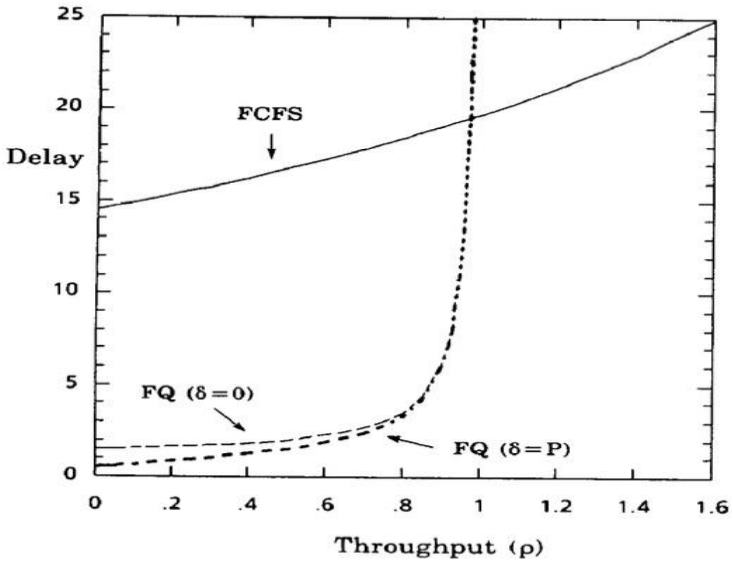


Figure 1: Delay vs. Throughput

Under FCFS

1) Telnet just adds to the congestion caused by FTP conversations and suffers high and increasing delays as it tries to increase its throughput.

Under WFQ regime,

1. As long as the Telnet source is sending under its fair share, it will get low delay.
2. If it exceeds its fair share, its delays will skyrocket as the Telnet queues get filled up at routers and packets start getting dropped.

Formal version of this result called the Parekh-Gallager theorem
"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case" and

"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case"

The Parekh-Gallager theorm (informally): *"If everyone promises to stay under a particular transmission rate, and the network can support the sum of these transmission rates, and the routers run WFQ, then everyone's worst-case per-packet delay can be bounded."*

In general, WFQ incentivizes good congestion control because a sender's bad behavior can only affect that sender. This is the exact opposite of FCFS/FIFO.

WFQ Tradeoffs:

- If we keep implementing more and more intelligence in the routers, this means switching speeds are going to be slower. This is an obstacle to deployment, even though this is changing now days.

- In WFQ, we need to maintain separate queues and other state information for each flow. This can lead to very high storage requirements, which can lead to scalability problems. Ex-
    1. In the core of an ISP
       Few millions of flows => Storing per flow state is infeasible
    2. In a datacenter
       Few thousand flows (Active flows are even less)=> Storing per flow state is feasible

WFQ Algorithm:

Reference: Section 3 of http://web.mit.edu/6.829/www/2016/papers/fq-notes.pdf)

Nagle's Round-Robin algorithm:

- Service each flow for a time quantum and then move on to the next

Nagle's Round-Robin has following problems:

- It's unfair if one flow uses much larger packets than the other.
- If a packet arrives just after its turn, it needs to wait a while for the time quanta of each of the other flows before it can be transmitted.

Idealized model:

- Bit-by-bit round robin. Go one bit at a time from each flow.
- Unattainable since sending single bit with headers is hugely inefficient.

Approximate bit-by-bit round robin:

We define

- R: Number of rounds made in the round-robin service discipline up to time t. Essentially, one round is one cycle through all active queues sending one bit per flow.
- N: Number of active flows
- mu: linespeed; the gateway's outgoing line

Then, **dR/dt = mu / N**

- Derivative of R is proportional to mu and inversely proportional to number of flows
- N keeps varying as the number of active queues change which changes dr/dt.
- Computing R through the above equation is hard. We use approximate values for R

Calculating Finish Time for each packet in a queue:

For a packet belonging to a flow which arrives at the gateway,

T1 = Finish_Time (in rounds) of previous packet in same queue if queue is non-empty
T2 = Current round number R, if queue is empty

Start_Time = MAX (T1, T2)

Since we are approximating sending one bit per flow in one round,

Finish_Time = Start_Time + Length of packet

Once we have Finish_Time for each packet in every queue, we can implement the algorithm below

Flow Control Algorithm:

1) Algorithm tracks “Per Flow State”:
   - Finish Time
   - Queue (head and tail pointers)
2) Sends packet with smallest finish time to "catch up" with bit-by-bit model
3) Approximate R as the finish time of current packet in service.

The approximations used above,

- Chose earliest finish time packet
- Set R to finish time of current packet in service

don’t introduce too much difference in the worst case relative to bit-by-bit round robin.

Our packet-by-packet transmission algorithm is simply defined by the rule that, whenever a packet finishes transmission, the next packet sent is the one with the smallest value of ‘Finish_Time’. In a preemptive version of this algorithm, newly arriving packets whose ‘Finish_Time’ is smaller than that of the packet currently in transmission preempt the transmitting packet. Practically the nonpreemptive version is easier to implement but the preemptive algorithm (with resumptive service) is more tractable analytically. Over sufficiently long conversations, both of the packetized algorithms asymptotically approach the fair bandwidth allocation of the Bitwise RR scheme.

Historical Impact:

In 1995, an algorithm called Deficit Round Robin (a variant of Round Robin but with an important fix) was developed. Even though it was not as good as WFQ, it was good enough and SIMPLE to implement. This algorithm is being widely used in routers as of today.