Congestion Control in the Network

Brighten Godfrey
CS 538 Sept 17 2013
How TCP congestion control is broken
Efficiency

Tends to fill queues (adding latency)

Slow to converge (for short flows or links with high bandwidth\(\times\)delay product)

Loss \(\neq\) congestion

May not fully utilize bandwidth
Limitations of TCP CC

Fairness

Unfair to large-RTT flows (less throughput)

Unfair to short flows if ssthresh starts small

Equal rates isn’t necessarily “fair” or best

Vulnerable to selfish & malicious behavior

• TCP assumes everyone is running TCP!
Limitations of TCP CC

Fills queues: adds loss, latency

Slow to converge

Loss ≠ congestion

May not utilize full bandwidth

Unfair to large-RTT

Unfair to short flows

Is equal rates really “fair”? 

Vulnerable to selfishness
Fills queues: adds loss, latency
Slow to converge
Loss ≠ congestion
May not utilize full bandwidth
Unfair to large-RTT
Unfair to short flows
Is equal rates really “fair”? 
Vulnerable to selfishness

Hard to use only end-to-end information to find ‘right’ rate
Obvious solution: Get more info from network
Limitations of TCP CC

- Fills queues: adds loss, latency
- Slow to converge
- Loss ≠ congestion
- May not utilize full bandwidth
- Unfair to large-RTT
- Unfair to short flows
- Is equal rates really “fair”?
- Vulnerable to selfishness

Hard to use only end-to-end information to find ‘right’ rate

Obvious solution: Get more info from network

Incentive issues
Congestion control with help from the network
Random early detection (RED)

- Drops more packets (randomly) as congestion increases
- Mechanism is entirely within routers

Explicit Congestion Notification (ECN)

- Mark bit in header instead of dropping

But what does the source really want?

- Just tell me the right rate, already!
- eXplicit Control Protocol (XCP)
- Rate Control Protocol (RCP)
A third reason TCP flows last so long is because of buffer queueing. In congestion-avoidance mode, TCP adapts slowly because of the additive increase, multiplicative decrease (AIMD) phase. First, it takes slow-start several round-trip times to finish an order of magnitude faster than for TCP. There are a variety of network conditions and traffic models. The value representative of traffic over a backbone link today, and these conditions (explained in the caption) were chosen to be a capacity = 2.4 Gbps, Round Trip Time = 100 ms, offered load = 0.25 packets/RTT.

Once a flow is in the AIMD phase, it is slow in catching the correct rate. Second, once a flow has reached the fair-share rate. In many cases, the flow has finished before TCP has found the correct rate. For a given network load, there are a number of active flows at any one time than there would be if they all were capable of finishing within one/few round-trip times – increases its rate until it encounters a loss, then experiences long flow durations. This is illustrated by AIMD results in long flow durations. This is illustrated with any spare capacity. Slow-start plus slow adaption exponential growth in numbers of flows in the system evolves like in figure 3. The steady-state number of flows equals \( \frac{L}{C} \) where \( L \) is the offered load, and \( C \) the link capacity. TCP deliberately fills the buffer at the bottleneck, thus the link utilization.

So why do TCP and XCP result in such long flow durations? In PS, both flows would complete in one round-trip time, but TCP is stretched over multiple round-trip times, which works well when all flows are long-lived. But as our plots show, in a dynamic environment, there are more flows in progress relative to ideal PS, and so there are more flows in progress at any instant.

TCP and XCP, when compared to ideal PS, we used simple deterministic examples to clarify each factor. Figure 2 shows an example with TCP and XCP; the top plot compares the mean flow durations, the middle plot shows the number of active flows with time and the bottom plot shows the link occupancy. TCP deliberately fills the buffer at the bottleneck, thereby increasing the duration of each flow even further. XCP can increase the duration of each flow even further because of a simple deterministic example in Figure 3. In the example, the link capacity is split equally between the two flows, and therefore operate below – often well below – their fair-share rate. Because of this, for a given network load, there are more active flows at any time than there would be if they are capable of finishing within one/few round-trip times.

Figure 2 shows an example with TCP and XCP; the top plot shows the average flow duration versus flow size, the middle plot shows the number of active flows versus time. In both cases, XCP results in longer durations and higher numbers of active flows. The bottom plot shows the number of active flows versus time. In both cases, XCP results in longer durations and higher numbers of active flows. The bottom plot shows the number of active flows versus time. In both cases, XCP results in longer durations and higher numbers of active flows.
Many flows waiting

[Fig. 1. The top plot shows the average flow duration versus flow size under various network conditions. The bottom plot shows the number of active flows versus time. In both plots, XCP can increase the duration of each flow even further compared to TCP and PS, consequently the flow duration is six times higher as well.]

[Dukkipati & McKeown ’05]

There are several reasons for the long duration of flows with TCP. First, it takes “slow-start” several round-trip times to find the correct rate. Second, once a flow has reached the “congestion-avoidance” mode, TCP adapts slowly because of additive increase, multiplicative decrease (AIMD) phase. While this was a deliberate choice to help stabilize TCP, it has the effect of increasing flow duration. Third, TCP can experience congestion-avoidance, short-lived flows never leave slow-start, the FCT for a flow of size \( p \) is \( \log_2(p) \times T \) RTTs, and the link utilization would be 100%. With TCP slow-start, the number of flows in system evolves like in figure 3. The steady-state number of flows equals 

\[
2^{\frac{T\text{RTT}}{2}} - 1
\]

where \( T \text{RTT} \) is the round trip time. Two flows, each of size 1000 packets, would complete in one RTT, while four flows, each of size 5000 packets, would complete in one RTT with XCP. Two flows, each of size 1000 packets, would complete in one RTT with PS, consequently the flow duration is six times higher as well. In PS, both flows would complete in one RTT, while two flows, each of size 5000 packets, would complete in one RTT with XCP. Two flows, each of size 1000 packets, would complete in one RTT with TCP, while four flows, each of size 5000 packets, would complete in one RTT with XCP.

The value of the flow size under XCP is much lower than under TCP due to an increase in the duration for XCP. For example, the link capacity is 2.4 Gbps, Round Trip Time = 100 ms, offered load = 0.15, availability = 1.4, link-utilization = 30%, the number of active flows at any one time is 2.5 times the number of flows if they are capable of finishing within one/few RTTs, and therefore operate below – often well below – their fair-share of link capacity. Slow-start plus slow adaption prolong flows unnecessarily. There seem to be four main reasons: (1) Flows start too slowly and are therefore artificially stretched to last many Round Trip Times (RTT) even if they are capable of finishing within one/few RTTs, (2) Bandwidth is shared rate. Because of this, for a given network load, there are several active flows at any one time than there would be if they were capable of finishing within one/few RTTs, (3) Buffers are filled (TCP) and therefore delay all flows, and (4) Timeouts and retransmissions due to packet losses (TCP). We will examine each reason in turn, and use simple examples to clarify each factor.

In this section, we will try to explain why both mechanisms, TCP and XCP, when compared to ideal PS, we used PS values are computed from analytical expressions [11].

So why do TCP and XCP result in such long flow durations? To illustrate how much longer flows take to complete with TCP and XCP from a simulation with Poisson flow arrivals, flow size under Pareto distributed with mean = 30 packets (1000 byte/pkt) and shape parameter = 1.4, link-utilization = 30%, the number of active flows at any one time is 2.5 times the number of flows if they are capable of finishing within one/few RTTs, and therefore operate below – often well below – their fair-share of link capacity. Slow-start plus slow adaption prolong flows unnecessarily. There seem to be four main reasons: (1) Flows start too slowly and are therefore artificially stretched to last many Round Trip Times (RTT) even if they are capable of finishing within one/few RTTs, (2) Bandwidth is shared rate. Because of this, for a given network load, there are several active flows at any one time than there would be if they were capable of finishing within one/few RTTs, (3) Buffers are filled (TCP) and therefore delay all flows, and (4) Timeouts and retransmissionsDue to packet losses (TCP). We will examine each reason in turn, and use simple examples to clarify each factor.

In this section, we will try to explain why both mechanisms, TCP and XCP, when compared to ideal PS, we used PS values are computed from analytical expressions [11].

So why do TCP and XCP result in such long flow durations? To illustrate how much longer flows take to complete with TCP and XCP from a simulation with Poisson flow arrivals, flow size under Pareto distributed with mean = 30 packets (1000 byte/pkt) and shape parameter = 1.4, link-utilization = 30%, the number of active flows at any one time is 2.5 times the number of flows if they are capable of finishing within one/few RTTs, and therefore operate below – often well below – their fair-share of link capacity. Slow-start plus slow adaption prolong flows unnecessarily. There seem to be four main reasons: (1) Flows start too slowly and are therefore artificially stretched to last many Round Trip Times (RTT) even if they are capable of finishing within one/few RTTs, (2) Bandwidth is shared rate. Because of this, for a given network load, there are several active flows at any one time than there would be if they were capable of finishing within one/few RTTs, (3) Buffers are filled (TCP) and therefore delay all flows, and (4) Timeouts and retransmissions due to packet losses (TCP). We will examine each reason in turn, and use simple examples to clarify each factor.
Rate Control Protocol [Dukkipati, Kobayashi, Zhang-Shen, McKeown, IWQoS 2005]

Router’s algorithm:

• Compute fair per-flow rate $R(t)$ at time $t$ as whatever will fill up the link capacity (roughly)
• Tell end-hosts about this by putting the value in packets, and recompute every RTT
RCP rate computation

\[ R(t) = R(t - d_0) + \frac{\alpha(C - y(t)) - \beta \frac{q(t)}{d_0}}{\hat{N}(t)} \]

**Simpler than XCP:**

- rates instead of windows
- thus, feedback doesn’t depend on a flow’s RTT
- thus, same feedback to everyone

(How can you estimate # flows?)
Estimating the number of flows

\[ \hat{N}(t) = \frac{C}{R(t - d_0)} \]

If guess is wrong, what happens?

- Queue builds up; will reduce rate in next round
- Possibly this estimator could be improved
RCP finishes flows quickly

The graph shows the average flow completion time (in seconds) for different flow sizes. The x-axis represents the flow size in packets, ranging from 0 to 2000 packets. The y-axis represents the average flow completion time in seconds, ranging from 0.1 to 100 seconds.

The graph includes lines and markers for various conditions:
- **XCP (avg.)** (gray dashed line with gray markers)
- **TCP (avg.)** (green solid line with green markers)
- **RCP (avg.)** (blue dotted line with blue markers)
- **Slow-Start** (red solid line with red markers)
- **PS** (orange dotted line with orange markers)

The RCP line is consistently lower than the TCP and XCP lines, indicating that RCP completes flows more quickly. The figure also shows that the maximum delay for RCP is often ten times the mean delay.

For any fixed simulation time, RCP was better than TCP and XCP for flows that completed, but it also finished more flows (and more work) than TCP and XCP.

With longer flows (e.g., 2000 pkts), the ratio of XCP and RCP delay still remains around 30, while TCP and RCP are similar.

Note that in RCP, the maximum delay experienced by the flows is also very close to the average delay for a given flow size.
Enforcing fairness and isolation

Based on slides by Ion Stoica
Problem: no isolation across flows

Assume router uses First In First Out (FIFO) queue

No protection: if a flow misbehaves it will hurt the other flows

Example: 1 UDP (10 Mbps) and 31 TCP’s sharing a 10 Mbps link
Round robin among different flows [Nagle ’87]

- One queue per flow
- while (1) { send one packet from each queue }
Advantages: protection among flows

- Misbehaving flows will not affect the performance of well-behaving flows
- FIFO does not have such a property

Disadvantages:

- More complex than FIFO: per flow queue/state
- Biased toward large packets: a flow receives service proportional to the number of packets
Define a **fluid flow** system: a system in which flows are served continuously

- essentially, **bit-by-bit round robin**

**Advantages**

- Each flow will receive exactly its max-min fair rate
- ...and exactly its fair per-packet delay
- ...regardless of packet sizes
If link congested, compute $f$ such that

$$\sum_i \min(r_i, f) = C$$

For $f = 4$:

- $\min(8, 4) = 4$
- $\min(6, 4) = 4$
- $\min(2, 4) = 2$
What we just saw was bit-by-bit round robin

But can’t interrupt transfer of a packet (why not?)

Idea: serve packets in the order in which they would have finished transmission in the fluid flow system

Strong guarantees: same as having a virtual link of the max-min fair capacity. Each flow gets:

- Exactly its max-min fair rate (+/- one packet size)
- Exactly its max-min fair per-packet delay (+/- one packet size) or better
Example

Flow 1 (arrival traffic)

Flow 2 (arrival traffic)

Service in fluid flow system

Packet system
Problem

Recall: “serve packets in the order in which they would have finished transmission in the fluid flow system”

So, need to compute finish time of each packet in the fluid flow system

... but new packet arrival can change finish times of existing packets (perhaps all)!

Updating those times would be expensive

Solution: *virtual time*
Key Observation: finish times may change when a new packet arrives, but the finish order doesn’t

- Only the order is important for scheduling

Solution: maintain the number of rounds needed to send the remaining bits of the packet

- New packet arrival doesn’t change # remaining rounds
- Does change rounds executed per unit time, but that’s ok

System virtual time = index of the final round in the bit-by-bit round robin scheme
System Virtual Time: $V(t)$

Measure service, instead of time

Slope of $V(t)$ = rate at which every active flow receives service

- $C =$ link capacity
- $N(t) =$ number of active flows in fluid flow system at time $t$

\[
\frac{\partial V(t)}{\partial t} = \frac{C}{N(t)}
\]
Define

- $F_{i}^{k}$ = virtual finishing time of packet $k$ of flow $i$
- $a_{i}^{k}$ = arrival time of packet $k$ of flow $i$
- $L_{i}^{k}$ = length of packet $k$ of flow $i$

Virtual finishing time of packet $k+1$ of flow $i$ is

$$F_{i}^{k+1} = \max(V(a_{i}^{k}), F_{i}^{k}) + L_{i}^{k+1}$$

Order packets by increasing virtual finishing time, and send them in that order
Weighted Fair Queueing (WFQ)

What if we don't want exact fairness?

• Maybe web traffic is more important than file sharing

Assign weight $w_i$ to each flow $i$

And change virtual finishing time to

$$F_{i}^{k+1} = \max(V(a_{i}^{k}), F_{i}^{k}) + \frac{L_{i}^{k+1}}{w_{i}}$$
FQ summary

FQ does not eliminate congestion; it just manages the congestion

Provides isolation between flows

- complete isolation?

Still need both end-host and router-based congestion control

- End-host congestion control to adapt rate
- Router congestion control to protect/isolate
Rethinking “fairness”: Congestion pricing
The Internet routes money; packets are just a side effect.

– Unknown, via Dave Clark
What is “fair”?

Flow rate equality!

Easily circumvented

Doesn’t even optimize for any metric of interest

Fig. 1: Poppycock.
Fairness for real life resources

Plentiful: use as much as you want
- air
- advisor’s grant money

Scarce: pay for what you want
- price set by market
- result (under assumptions): socially optimal allocation

Fig. 2: Invisible hand of the market.
Briscoe’s main points

Flow rate fairness (FRF) is not useful

Cost fairness is useful

Flow rate fairness is hard to enforce

Cost fairness is feasible to enforce

Fig 3: Briscoe.
FRF not useful

Doesn’t equalize benefits

- e.g., SMS message vs. a packet of a video stream

Doesn’t equalize costs

- e.g., “parking lot” network: long flow causes significant congestion but is given equal rate by fair queueing

Therefore, doesn’t equalize cost or benefit
Myopic: no notion of fairness across time

In summary, FRF does not optimize utility

- except for strange definitions of utility...

So, even cooperating entities should not use it!
Cost fairness is useful

Economic entities pay for the costs they incur

- This is “fair” (in a real-world sense), not “equal”—and that’s fine

In other words, networks charge packets for the congestion they cause

- Can networks lie about congestion?
- Yes. So it’s really a market price, not exactly congestion

Result: senders want to maximize utility

- Will balance benefit with cost (*utility* = benefit – cost)
Example: light & heavy traffic

[Briscoe 2009]

Key point: Benefit per bit is high for light flow and low for heavy flow.
Frank Kelly 1997: Cost fairness maximizes aggregate utility

i.e.: any different outcome results in suboptimal utility

Why won’t anyone listen to Kelly? Hello?! ... where did everybody go?
Each user $i$ has utility $U_i(r_i)$ for rate $r_i$

Each user $i$ pays $p_i$ for access to link (its own choice)

Link sets price per unit bandwidth: $p = (\text{Sum } p_j) / C$

- thus, $r_i = p_i / p = C p_i / (\text{Sum } p_j)$

Theorem: assuming $U_i$ concave, strictly increasing, and continuously differentiable, then

- A competitive equilibrium exists: setting of $p_i$s in which no user can improve their utility given current price
- This equilibrium maximizes $\text{Sum } U_i(r_i)$
Run your flow longer

Create more flows (similar to sybil attack)

- Multiple TCP connections between same source/destination (web browsers)
- Spoof source IP / MAC address
- Multiple flows to other destinations (BitTorrent)
Cost fairness is enforceable

You send me a packet; I handle delivery and charge you for it

How much do I charge?

• Depends on cost on entire remainder of path!

Not the only way of arranging payments, but it is convenient

• payments are between neighbors that already have an economic relationship
Mechanism: Re-Feedback

Key property: every hop knows total congestion along downstream path

First packet

Second packet
Previous explanation was in terms of money, but doesn’t have to directly involve money

- Re-feedback is a mechanism
- Doesn’t imply a particular way of implementing congestion pricing

Possible variants of congestion pricing

- pay per packet?
- monthly allowance?
- only at edges?
- between all ISPs?
Host running a persistent “light” job is interrupted by heavy flows congesting the net?

Host is compromised? (botnet) Who pays?

If we want cost fairness, is Weighted Fair Queueing useless?

- No: provides mechanism to isolate flows, virtualize links
- e.g., could use congestion pricing to set WFQ’s weights
"It just isn’t realistic to create a system the size of the Internet and define fairness within the system without reference to fairness outside the system."

Cost fairness optimizes aggregate utility and is feasible to enforce.

Flow rate fairness does not optimize utility and is not feasible to enforce.

- Cease publication on the topic and stop teaching it in undergraduate courses.
Announcements
Announcements

Assignment 1 was due 2pm today

- Accepting late submissions (-15%) till 2pm Wed
- No credit thereafter

A bunch more project ideas

- To be released late tonight (see Piazza)

Next reading: Forwarding hardware