Rate adaptation, Congestion Control and Fairness: A Tutorial

JEAN-YVES LE BOUDEC
Ecole Polytechnique Fédérale de Lausanne (EPFL)

Available at `https://leboudec.github.io/leboudec/resources/tutorial.html`

November 1, 2021

Thanks to all who contributed to reviews and debugging of this document, in particular B. Radunovic, M. Vojnovic and Miroslav Popovic.

Contents

CHAPTER 1

CONGESTION CONTROL FOR BEST EFFORT: THEORY

In this chapter you learn

- why it is necessary to perform congestion control
- the additive increase, multiplicative decrease rule
- the three forms of congestion control schemes
- max-min fairness, proportional fairness

This chapter gives the theoretical basis required for Congestion Control in the Internet.

1.1 THE OBJECTIVE OF CONGESTION CONTROL

1.1.1 CONGESTION COLLAPSE

Consider a network where sources may send at a rate limited only by the source capabilities. Such a network may suffer of congestion collapse, which we explain now on an example.

We assume that the only resource to allocate is link bit rates. We also assume that if the offered traffic on some link l exceeds the capacity c_l of the link, then all sources see their traffic reduced in proportion of their offered traffic. This assumption is approximately true if queuing is first in first out in the network nodes, neglecting possible effects due to traffic burstiness.

Consider first the network illustrated on Figure 1.1. Sources 1 and 2 send traffic to destination nodes D1 and D2 respectively, and are limited only by their access rates. There are five links labeled 1 through 5 with capacities shown on the figure. Assume sources are limited only by their first link, without feedback from the network. Call λ_i the sending rate of source i, and $\lambda' i$ the outgoing rate.

For example, with the values given on the figures we find $\lambda_1 = 100$kb/s and $\lambda_2 = 1000$kb/s, but only $\lambda'_1 = \lambda'_2 = 10$kb/s, and the total throughput is 20kb/s ! Source 1 can send only at 10 kb/s

because it is competing with source 2 on link 3, which sends at a high rate on that link; however, source 2 is limited to 10 kb/s because of link 5. If source 2 would be aware of the global situation, and if it would cooperate, then it would send at 10 kb/s only already on link 2, which would allow source 1 to send at 100 kb/s, without any penalty for source 2. The total throughput of the network would then become $\theta = 110 kb/s$.

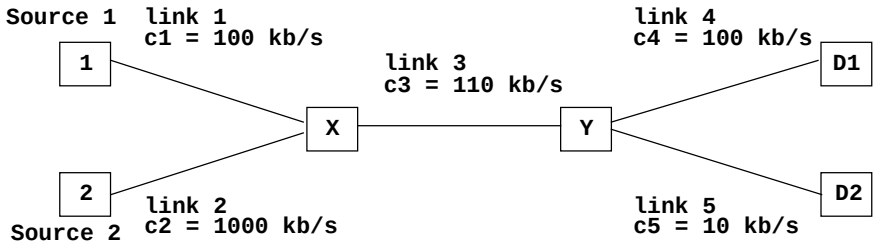


Figure 1.1: A simple network exhibiting some inefficiency if sources are not limited by some feedback from the network

The first example has shown some inefficiency. In complex network scenarios, this may lead to a form of instability known as congestion collapse. To illustrate this, we use the network illustrated on Figure 1.2. The topology is a ring; it is commonly used in many networks, because it is a simple way to provide some redundancy. There are I nodes and links, numbered $0, 1, ..., I-1$. Source i enters node i, uses links $[(i+1) \bmod I]$ and $[(i+2) \bmod I]$, and leaves the network at node $(i+2) \bmod I$. Assume that source i sends as much as λ_i, without feedback from the network. Call λ'_i the rate achieved by source i on link $[(i+1) \bmod I]$ and λ''_i the rate achieved on link $[(i+2) \bmod I]$. This corresponds to every source choosing the shortest path to the destination. In the rest of this example, we omit "$\bmod I$" when the context is clear. We have then:

$$\begin{cases} \lambda'_i = \min\left(\lambda_i, \frac{c_i}{\lambda_i + \lambda'_{i-1}} \lambda_i\right) \\ \lambda''_i = \min\left(\lambda'_i, \frac{c_{i+1}}{\lambda'_i + \lambda_{i+1}} \lambda'_i\right) \end{cases} \tag{1.1}$$

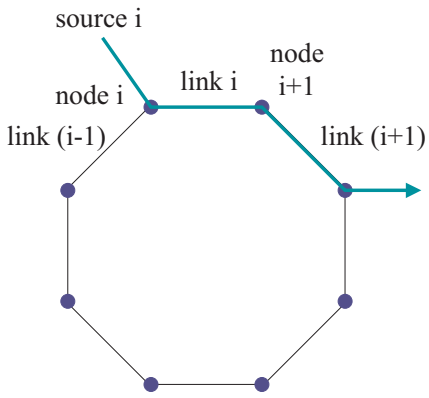


Figure 1.2: A network exhibiting congestion collapse if sources are not limited by some feedback from the network

Applying Equation 1.1 enables us to compute the total throughput θ. In order to obtain a closed form solution, we further study the symmetric case, namely, we assume that $c_i = c$ and $\lambda_i = \lambda$ for

all i. Then we have obviously $\lambda'_i = \lambda'$ and $\lambda''_i = \lambda''$ for some values of λ' and λ'' which we compute now.

If $\lambda \leq \frac{c}{2}$ then there is no loss and $\lambda'' = \lambda' = \lambda$ and the throughput is $\theta = I\lambda$. Else, we have, from Equation (1.1)

$$\lambda' = \frac{c\lambda}{\lambda + \lambda'}$$

We can solve for λ' (a polynomial equation of degree 2) and obtain

$$\lambda' = \frac{\lambda}{2}\left(-1 + \sqrt{1 + 4\frac{c}{\lambda}}\right)$$

We have also from Equation (1.1)

$$\lambda'' = \frac{c\lambda'}{\lambda + \lambda'}$$

Combining the last two equations gives

$$\lambda'' = c - \frac{\lambda}{2}\left(\sqrt{1 + 4\frac{c}{\lambda}} - 1\right)$$

Using the limited development, valid for $u \to 0$

$$\sqrt{1+u} = 1 + \frac{1}{2}u - \frac{1}{8}u^2 + o(u^2)$$

we have

$$\lambda'' = \frac{c^2}{\lambda} + o(\frac{1}{\lambda})$$

Thus, the limit of the achieved throughput, when the offered load goes to $+\infty$, is 0. This is what we call *congestion collapse*.

Figure 1.3 plots the throughput per source λ'' as a function of the offered load per source λ. It confirms that after some point, the throughput decreases with the offered load, going to 0 as the offered load goes to $+\infty$.

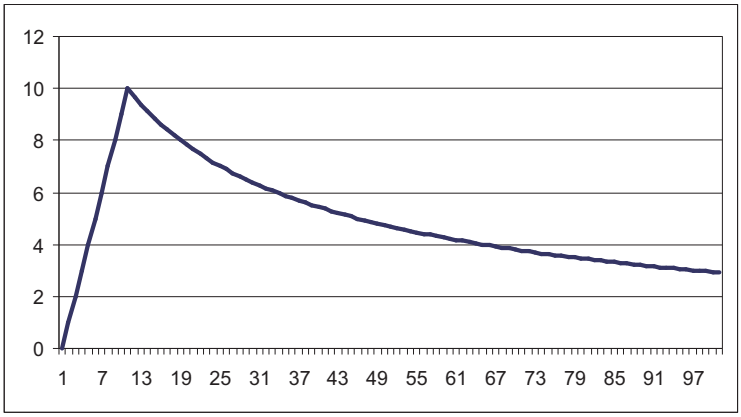


Figure 1.3: Throughput per source as a function of the offered load per source, in Mb/s, for the network of Figure 1.2. Numbers are in Mb/s. The link rate is $c = 20$Mb/s for all links.

The previous discussion has illustrated the following fact:

FACT 1.1.1 (Efficiency Criterion). *In a packet network, sources should limit their sending rate by taking into consideration the state of the network. Ignoring this may put the network into congestion collapse. One objective of congestion control is to avoid such inefficiencies.*

Congestion collapse occurs when some resources are consumed by traffic that will be later discarded. This phenomenon did happen in the Internet in the middle of the eighties. At that time, there was no end-to-end congestion control in TCP/IP. As we will see in the next section, a secondary objective is fairness.

1.1.2 EFFICIENCY VERSUS FAIRNESS

Assume that we want to maximize the network throughput, based on the considerations of the previous section. Consider the network example in Figure 1.4; one source sends at rate x_0Mb/s, one at rate x_1Mb/s and nine sources at rate x_2Mb/s each. We assume that we implement some form of congestion control and that there are negligible losses. Thus, the flow, in Mb/s, on the first link i is x_0+x_1, and on the second link it is x_0+9x_2. For a given value of x_0, maximizing the throughput requires that $x_1 = 10-x_0$ and $9x_2 = 10-x_0$. The total throughput, measured at the network output, is then $x_0+x_1+9x_2 = 20-x_0$; it is maximum for $x_0 = 0$!

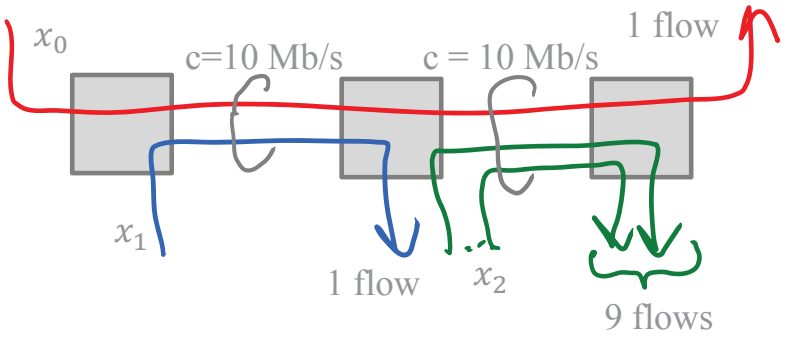


Figure 1.4: A simple network used to illustrate fairness and efficiency.

The example shows that maximizing network throughput as a primary objective may lead to gross unfairness; in the worst case, some sources may get a zero throughput, which is probably considered unfair by these sources.

In general, the concept of efficiency is captured by the notion of *Pareto Efficiency*. Consider an allocation problem; define the vector $\vec{x}$ whose ith coordinate is the allocation for user i.

DEFINITION 1.1.1 (Pareto Efficiency). *A feasible allocation of rates $\vec{x}$ is "Pareto-Efficient" (also called "Pareto-Optimal") if and only if an increase of any rate within the domain of feasible allocations must be at the cost of a decrease of some other rate. Formally, for any other feasible allocation $\vec{y}$, if $y_s > x_s$ then there must exist some s' such that $y_{s'} < x_{s'}$.*

In general, there exist many Pareto-efficient allocations. For the example in Figure 1.4, any allocation that saturates every link (i.e., such that $x_0+x_1 =$ and $x_0+9x_2 = 10$) is Pareto-efficient. The allocation that maximizes total throughput (and has $x_0 = 0$) is one of them; another Pareto efficient allocation is $x_0 = 0.1, x_1 = 9.9, x_2 = 1.1$; yet another one is $x_0 = 1, x_1 = 9, x_2 = 1$. Among all of

these allocations, which one should a fair and efficient congestion control scheme choose ? This requires a formal definition of fairness, given in the next section.

1.2 FAIRNESS

1.2.1 MAX-MIN FAIRNESS

In an egalitarian vision, fairness would simply mean allocating the same share to all, while maximizing this share. In the simple case of Figure 1.4, this would mean allocating $x_0 = x_1 = x_2 = 1$. However, such an egalitarian allocation is not Pareto-efficient (because it is possible to unilaterally increase x_1).

A better allocation could be based on the following observation. Starting from the egalitarian allocation, we can increase the rate x_1 without impacting other rates, up to the value $x_1 = 9$. The resulting allocation $x_0 = 1, x_1 = 9, x_2 = 1$ *is* Pareto-efficient, and still appears to be as fair as possible. It turns out that this allocation is exactly the Max-Min fair allocation, which is defined next.

DEFINITION 1.2.1 (Max-min Fairness [2]). *A feasible allocation of rates $\vec{x}$ is "max-min fair" if and only if an increase of any rate within the domain of feasible allocations must be at the cost of a decrease of some already smaller rate. Formally, for any other feasible allocation $\vec{y}$, if $y_s > x_s$ then there must exist some s' such that $x_{s'} \leq x_s$ and $y_{s'} < x_{s'}$.*

The name "max-min" comes from the idea that it is forbidden to decrease the share of sources that have small values, thus, in some sense, we give priority to flows with small values.

THEOREM 1.2.1. *A max-min fair allocation is Pareto-efficient.*

PROOF: If we increase one rate in a max-min fair allocation, we must decrease some other rate, which expresses that the allocation is Pareto-efficient. □

The converse is not true – a Pareto-efficient allocation is not, in general, max-min fair. In fact, as we will see next, the max-min fair allocation is unique, whereas there are usually many Pareto-efficient allocations – see the in Figure 1.4.

THEOREM 1.2.2. *A max-min fair allocation, if it exists, is unique.*

PROOF: Assume now that $\vec{x}$ and $\vec{y}$ are two max-min fair allocations for the same problem, with $\vec{x} \neq \vec{y}$. Without loss of generality, we can assume that there exists some i such that $x_i < y_i$. Consider the smallest value of x_i that satisfies $x_i < y_i$, and call i_0 the corresponding index. Thus, $x_{i_0} < y_{i_0}$ and

$$\text{if } x_i < y_i \text{ then } x_{i_0} \leq x_i \tag{1.2}$$

Now since $\vec{x}$ is max-min fair, from Definition 1.2.1, there exists some j with

$$y_j < x_j \leq x_{i_0} \tag{1.3}$$

Now $\vec{y}$ is also max-min fair, thus by the same token there exists some k such that

$$x_k < y_k \leq y_j \tag{1.4}$$

Combining (1.3) and (1.4), we obtain

$$x_k < y_k \leq y_j < x_j \leq x_{i_0}$$

which contradicts (1.2). □ □

For some allocation problems, a max-min fair allocation might not exist. However, for the practical examples of interest in the context of congestion control, there is always one (and only one) max-min fair allocation, as we show next. For more general cases, see [23], where it is shown in particular that a max-min fair allocation always exists when the feasible set is convex and compact.

NETWORK MODEL We use the following simplified network model in the rest of this section. We consider a set of sources $s = 1,\ldots,S$ and links $1,\ldots,L$. Let $A_{l,s}$ be the fraction of traffic of source s which flows on link l, and let c_l be the capacity of link l. We define a network as the couple $(\vec{x},A)$.

A *feasible allocation* of rates $x_s \geq 0$ is defined by: $\sum_{s=1}^{S} A_{l,s}x_s \leq c_l$ for all l.

Our network model supports both multicast and load sharing. For a given source s, the set of links l such that $A_{l,s} > 0$ is the path followed by the data flow with source s. In the simplest case (no load sharing), $A_{l,s} \in \{0,1\}$; if a flow from source s is equally split between two links l_1 and l_2, then $A_{l_1,s} = A_{l_2,s} = 0.5$. In principle, $A_{l,s} \leq 1$, but this is not mandatory (in some encapsulation scenarios, a flow may be duplicated on the same link).

We now show that there exists a max-min fair allocation to our network model, and how to obtain it. This will result from the key concept of "bottleneck link".

DEFINITION 1.2.2 (Bottleneck Link). *With our network model above, we say that link l is a bottleneck for source s if and only if*

1. *link l is saturated: $c_l = \sum_i A_{l,i}x_i$*
2. *source s on link l has the maximum rate among all sources using link l: $x_s \geq x_{s'}$ for all s' such that $A_{l,s'} > 0$.*

Intuitively, a bottleneck link for source s is a link which is limiting, for a given allocation. In the previous example, the first link is a bottleneck for blue flow with rate x_1 and for the red flow with rate x_0; the second link is a bottleneck for each of the nine green flows with rates x_2 and for the red flow with rate x_0.

THEOREM 1.2.3. *A feasible allocation of rates $\vec{x}$ is max-min fair if and only if every source has a bottleneck link.*

PROOF: **Part 1.** Assume that every source has a bottleneck link. Consider a source s for which we can increase the rate x_s while keeping the allocation feasible. Let l be a bottleneck link for s. Since l is saturated, it is necessary to decrease $x_{s'}$ for some s' such that $A_{l,s'} > 0$. We assumed that we can increase the rate of s: thus there must exist some $s' \neq s$ that shares the bottleneck link l. But for all such s', we have $x_s \geq x_{s'}$, thus we are forced to decrease $x_{s'}$ for some s' such that $x_s \geq x_{s'}$: this shows that the allocation is max-min fair.

Part 2. Conversely, assume that the allocation is max-min fair. For any source s, we need to find a bottleneck link. We proceed by contradiction. Assume there exists a source s with no

bottleneck link. Call L_1 the set of saturated links used by source s, namely, $L_1 = \{l$ such that $c_l = \sum_i A_{l,i}x_i$ and $A_{l,s} > 0\}$. Similarly, call L_2 the set of non-saturated links used by source s. Thus a link is either in L_1 or L_2, or is not used by s. Assume first that L_1 is non-empty.

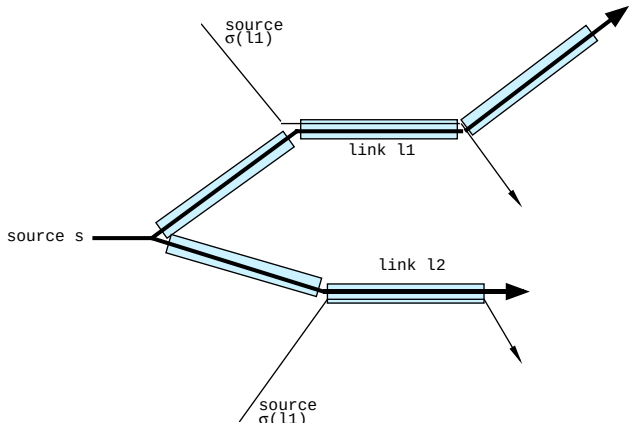


Figure 1.5: A network example showing one multicast source

By our assumption, for all $l \in L_1$, there exists some s' such that $A_{l,s'} > 0$ and $x_{s'} > x_s$. Thus we can build a mapping σ from L_1 into the set of sources $\{1,\ldots,S\}$ such that $A_{l,\sigma(l)} > 0$ and $x_{\sigma(l)} > x_s$ (see Figure 1.5 for an illustration). Now we will show that we can increase the rate x_s in a way that contradicts the max-min fairness assumption. We want to increase x_s by some value δ, at the expense of decreasing $x_{s'}$ by some other values $\delta_{s'}$, for all s' that are equal to some $\sigma(l')$. We want the modified allocation to be feasible; to that end, it is sufficient to have:

$$A_{l,s}\delta \leq A_{l,\sigma(l)}\delta_{\sigma(l)} \quad \text{for all } l \in L_1 \tag{1.5}$$

$$A_{l,s}\delta \leq c_l - \sum_i A_{l,i}x_i \quad \text{for all } l \in L_2 \tag{1.6}$$

$$\delta_{\sigma(l)} \leq x_{\sigma(l)} \quad \text{for all } l \in L_1 \tag{1.7}$$

Equation (1.5) expresses that the increase of flow due to source s on a saturated link l is at least compensated by the decrease of flow due to source $\sigma(l)$. Equation (1.6) expresses that the increase of flow due to source s on a non-saturated link l does not exceed the available capacity. Finally, equation (1.7) states that rates must be non-negative.

This leads to the following choice.

$$\delta = \min_{l\in L_1}\{\frac{x_{\sigma(l)}A_{l,\sigma(l)}}{A_{l,s}}\} \wedge \min_{l\in L_2}\{\frac{c_l - \sum_i A_{l,i}x_i}{A_{l,s}}\} \tag{1.8}$$

which ensures that Equation (1.6) is satisfied and that $\delta > 0$.

In order to satisfy Equations (1.5) and (1.7) we need to compute the values of $\delta_{\sigma(l)}$ for all l in L_1. Here we need to be careful with the fact that the same source s' may be equal to $\sigma(l)$ for more than one l. We define $\delta(s')$ by

$$\delta(s') \quad = 0 \text{ if there is no } l \text{ such that } s' = \sigma(l) \tag{1.9}$$

$$\delta(s') \quad = \max_{\{l \text{ such that } \sigma(l)=s'\}}\{\frac{\delta A_{l,s}}{A_{l,\sigma(l)}}\} \text{ otherwise} \tag{1.10}$$

This definition ensures that Equation (1.5) is satisfied. We now examine Equation (1.7). Consider some s' for which there exists an l with $\sigma(l) = s$, and call l_0 the value which achieves the maximum in (1.10), namely:

$$\delta(s') = \frac{\delta A_{l_0,s}}{A_{l_0,s'}} \tag{1.11}$$

From the definition of δ in (1.8), we have

$$\delta \leq \frac{x_{\sigma(l_0)} A_{l_0,\sigma(l_0)}}{A_{l_0,s}} = \frac{x_{s'} A_{l_0,s'}}{A_{l_0,s}}$$

Combined with (1.11), this shows that Equation (1.7) holds. In summary, we have shown that we can increase x_s at the expense of decreasing the rates for only those sources s' such that $s' = \sigma(l)$ for some l. Such sources have a rate higher than x_s, which shows that the allocation $\vec{x}$ is not max-min fair and contradicts our hypothesis.

It remains to examine the case where L_1 is empty. The reasoning is the same, we can increase x_s without decreasing any other source, and we also have a contradiction. □

THE WATER-FILLING ALGORITHM The previous theorem is particularly useful in deriving a practical method for obtaining a max-min fair allocation, called "progressive filling" or "water filling". The idea is as follows. You start with all rates equal to 0 and grow all rates together at the same pace, until one or several link capacity limits are hit. The rates for the sources that use these links are not increased any more, and you continue increasing the rates for other sources. All the sources that are stopped have a bottleneck link. This is because they use a saturated link, and all other sources using the saturated link are stopped at the same time, or were stopped before, thus have a smaller or equal rate. The algorithm continues until it is not possible to increase. The algorithm terminates because L and S are finite. Lastly, when the algorithm terminates, all sources have been stopped at some time and thus have a bottleneck link. By application of Theorem 1.2.3, the allocation is max-min fair.

EXAMPLE Let us apply the water-filling algorithm to Figure 1.4.

We first let $x_0 = x_1 = x_2 = t$ and increase t until we hit a limit, i.e. we maximize t subject to the constraints imposed by the link capacities. The constraints are

$$x_0 + x_1 \leq 10 \text{ and } x_1 + 9x_2 \leq 10$$

which gives

$$2t \leq 10 \text{ and } 10t \leq 10$$

The maximum is for $t = 1$Mb/s and at this value the second constraint is hit, i.e., the second link is a bottleneck link. The sources using this link are the sources of type 0 and 2, therefore we let

$$x_0 = 1 \text{ and } x_2 = 1$$

and these values are final.

In a second round we increase the rates of the other sources, namely we increase x_1 until a capacity limit is hit. There is only one constraint left, that of the first link, which is now expressed as

$$1 + x_1 \leq 10$$

The maximal value of x_1 is 9. With this, all sources are stopped and the algorithm terminates. The max-min fair allocation is thus $x = 1, x_1 = 9, x_2 = 1$.

We see that all sources of type 0 and 2 obtain the same rate. In some sense, max-min fairness ignores the fact that sources of type 0 use more network resources than those of type 2.

THEOREM 1.2.4. *For the network defined above, with fixed routing parameters $A_{l,s}$, there exists a unique max-min fair allocation. It can be obtained by the water-filling algorithm.*

PROOF: We have already proven uniqueness. Existence follows from the water-filling algorithm. □

The notion of max-min fairness can be generalized by using weights in the definition [2, 18].

1.2.2 PROPORTIONAL FAIRNESS

The previous definition of fairness puts emphasis on maintaining high values for the smallest rates. As shown in the previous example, this may be at the expense of some network inefficiency. An alternative definition of fairness has been proposed in the context of game theory [22].

DEFINITION 1.2.3 (Proportional Fairness). *An allocation of rates $\vec{x}$ is "proportionally fair" if and only if all rates are positive and, for any other feasible allocation $\vec{y}$, we have:*

$$\sum_{s=1}^{S} \frac{y_s - x_s}{x_s} \leq 0$$

In other words, any change in the allocation must have a negative average change.

Let us consider for example the parking lot scenario in Figure 1.6. Is the max-min fair allocation proportionally fair ?

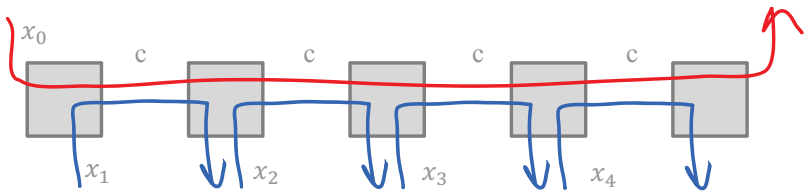


Figure 1.6: A simple network used to illustrate proportional fairness (the"parking lot" scenario)

To get the answer, observe that the max-min fair allocation is obtained easily with water-filling and is $x_s = c/2$ for $s = 0, 1..4$. Consider a new allocation resulting from a decrease of x_0 equal to δ:

$$\begin{aligned} y_0 &= \tfrac{c}{2} - \delta \\ y_s &= \tfrac{c}{2} + \delta \; s = 1, \ldots, 4 \end{aligned}$$

For $\delta < \frac{c}{2}$, the new allocation $\vec{y}$ is feasible. The average rate of change is

$$\left(\sum_{s=1}^{4} \frac{2\delta}{c} \right) - \frac{2\delta}{c} = \frac{6\delta}{c}$$

which is positive. Thus the max-min fair allocation for this example is not proportionally fair. In this example, we see that a decrease in rate for sources of type 0 is less important than the corresponding increase which is made possible for the other sources, because the increase is multiplied by the number of sources. Informally, we say that proportional fairness takes into consideration the usage of network resources.

THEOREM 1.2.5. *A proportionally fair allocation is Pareto-efficient.*

PROOF: Let $\vec{x}$ be a proportionally fair allocation and let $\vec{y}$ be some feasible allocation such that $y_i > x_i$ for some i. The rate of change is

$$\frac{y_i - x_i}{x_i} + \sum_{s=1\ldots S, s\neq i} \frac{y_s - x_s}{x_s} \leq 0$$

because $\vec{x}$ is proportionally fair. The first term is positive, therefore at least one of the other terms, say $\frac{y_s - x_s}{x_s}$ must be negative, and thus $y_s < x_s$. This shows that $\vec{x}$ is Pareto-efficient. □

Now we derive a practical result which can be used to compute a proportionally fair allocation. To that end, we interpret the average rate of change as $\nabla J_{\vec{x}} \cdot (\vec{y} - \vec{x})$, with

$$J(\vec{x}) = \sum_s \ln(x_s)$$

Thus, intuitively, a proportionally fair allocation should maximize J.

THEOREM 1.2.6. *There exists one unique proportionally fair allocation. It is obtained by maximizing $J(\vec{x}) = \sum_s \ln(x_s)$ over the set of feasible allocations.*

PROOF: 1. We first prove that the maximization problem has a unique solution. Function J is concave, as a sum of concave functions. The feasible set is convex, as intersection of convex sets, thus any local maximum of J is an absolute maximum. Now J is strictly concave, which means that

$$\text{if } 0 < \alpha < 1 \text{ then } J(\alpha\vec{x} + (1-\alpha)\vec{y}) > \alpha J(\vec{x}) + (1-\alpha)J(\vec{y})$$

This can be proven by studying the second derivative of the restriction of J to any linear segment. Now a strictly concave function has at most one maximum on a convex set.

Now J is continuous if we allow $\log(0) = -\infty$ and the set of feasible allocations is compact (because it is a closed, bounded subset of $\mathbb{R}^S$). Thus J has at least one maximum over the set of feasible allocations. Combining all the arguments together proves that J has exactly one maximum over the set of feasible allocations, and that any local maximum is also exactly the global maximum.

2. For any $\vec{\delta}$ such that $\vec{x} + \vec{\delta}$ is feasible,

$$J(\vec{x} + \vec{\delta}) - J(\vec{x}) = \nabla J_{\vec{x}} \cdot \vec{\delta} + \frac{1}{2}\vec{\delta}^T \nabla^2 J_{\vec{x}} \vec{\delta} + o(||\vec{\delta}||^2)$$

Now by the strict concavity, $\nabla^2 J_{\vec{x}}$ is definite negative thus, for $||\vec{\delta}||$ small enough:

$$\frac{1}{2}\vec{\delta}^T \nabla^2 J_{\vec{x}} \vec{\delta} + o(||\vec{\delta}||^2) < 0$$

and therefore

$$J(\vec{x}+\vec{\delta})-J(\vec{x}) \leq \nabla J_{\vec{x}} \cdot \vec{\delta}$$

Now assume that $\vec{x}$ is a proportionally fair allocation. This means that

$$\nabla(J)_{\vec{x}} \cdot \vec{\delta} \leq 0$$

and thus J has a local maximum at $\vec{x}$, thus also a global maximum. This also shows the uniqueness of a proportionally fair allocation.

3. Conversely, assume that J has a global maximum at $\vec{x}$. Observe that our network model assumes that there exists some positive and feasible allocation, therefore, the maximum of J is not $-\infty$, therefore the maximiser $\vec{x}$ satisfies $x_s > 0$ for all s. Let $\vec{y}$ be some other feasible allocation and call D the average rate of change. We also have:

$$D = \nabla(J)_{\vec{x}} \cdot (\vec{y}-\vec{x})$$

Since the feasible set is convex, the segment $[\vec{x},\vec{y}]$ is entirely feasible, and thus $J(\vec{x}+t(\vec{y}-\vec{x})) \leq J(\vec{x})$ for $t \in [0;1]$. Observe that

$$D = \lim_{t \to 0^+} \frac{J(\vec{x}+t(\vec{y}-\vec{x}))-J(\vec{x})}{t}$$

thus $D \leq 0$. □

EXAMPLE Let us apply Theorem 1.2.6 to the parking lot scenario in Figure 1.6. For any choice of x_0, we should set x_i such that

$$x_0 + x_i = c, i = 1, \ldots, 4$$

otherwise we could increase x_i without affecting other values, and thus increase function J. The value of x_0 is found by maximizing $f(x_0)$, defined by

$$f(x_0) = \ln(x_0) + \sum_{i=1}^{4} \ln(c - x_0)$$

over the set $0 \leq x_0 \leq c$. The derivative of f is

$$f'(x_0) = \frac{1}{x_0} - \frac{4}{c-x_0}$$

After some algebra, we find that the maximum is for

$$x_0 = \frac{c}{5}$$

and

$$x_i = \frac{4c}{5} \text{ for } i = 1...4$$

Compare with max-min fairness, where, in that case, the allocation is $\frac{c}{2}$ for all rates. We see that sources of type 0 get a smaller rate, since they use more network resources.

The concept of proportional fairness can easily extended to *weighted* proportional fairness, where the allocation maximizes a weighted sum of logarithms [14].

1.2.3 UTILITY APPROACH TO FAIRNESS

Proportional fairness is an example of a more general fairness concept, called the "utility" approach, which is defined as follows. Every source s has a utility function u_s where $u_s(x_s)$ indicates the value to source s of having rate x_s. Every link l (or network resource in general) has a cost function g_l, where $g_l(f)$ indicates the cost to the network of supporting an amount of flow f on link l. Then, a "utility fair" allocation of rates is an allocation which maximizes $H(\vec{x})$, defined by

$$H(\vec{x}) = \sum_{s=1}^{S} u_s(x_s) - \sum_{l=1}^{L} g_l(f_l)$$

with $f_l = \sum_{s=1}^{S} A_{l,s} x_s$, over the set of feasible allocations.

Proportional fairness corresponds to $u_s = \ln$ for all s, and $g_l(f) = 0$ for $f < c_l$, $g_l(f) = +\infty$ for $f \geq c_l$. Rate proportional fairness corresponds to $u_s(x_s) = w_s \ln(x_s)$ and the same choice of g_l.

Computing utility fairness requires solving constrained optimization problems; a reference is [27].

1.2.4 MAX MIN FAIRNESS AS A LIMITING CASE OF UTILITY FAIRNESS

We show in this section that max-min fairness is the limiting case of a utility fairness. Indeed, we can define a set of concave, increasing utility functions f_m, indexed by $m \in \mathbb{R}^+$ such that the allocation $\vec{x^m}$ which maximizes $\sum_{i=1}^{I} f_m(x_i)$ over the set of feasible allocations converges towards the max-min fair allocation.

The proof is a correct version of the ideas expressed in [21] and later versions of it.

Let f_m be a family of increasing, differentiable and concave functions defined on $\mathbb{R}^+$. Assume that, for any fixed numbers x and $\delta > 0$,

$$\lim_{m \to +\infty} \frac{f'_m(x+\delta)}{f'_m(x)} = 0 \tag{1.12}$$

The assumption on f_m is satisfied if f_m is defined by

$$f_m(x) = c - g(x)^m$$

where c is a constant and g is a differentiable, decreasing, convex and positive function. For example, consider $f_m(x) = 1 - \frac{1}{x^m}$.

Define $\vec{x^m}$ the unique allocation which maximizes $\sum_{i=1}^{I} f_m(x_i)$ over the set of feasible allocations. Thus $\vec{x^m}$ is the fair allocation of rates, in the sense of utility fairness defined by f_m. The existence of this unique allocation follows from the same reasoning as for Theorem 1.2.6. Our result if the following theorem.

THEOREM 1.2.7. *The set of utility-fair allocation $\vec{x^m}$ converges towards the max-min fair allocation as m tends to $+\infty$.*

The rest of this section is devoted to the proof of the theorem. We start with a definition and a lemma.

DEFINITION 1.2.4. *We say that a vector $\vec{z}$ is an accumulation point for a set of vectors $\vec{x}_m$ indexed by $m \in \mathbb{R}^+$ if there exists a sequence $m_n, n \in \mathbb{N}$, with $\lim_{n \to +\infty} m_n = +\infty$ and $\lim_{n \to +\infty} \vec{x^{m_n}} = \vec{z}$.*

LEMMA 1.2.1. *If $\vec{x}^*$ is an accumulation point for the set of vectors $\vec{x^m}$, then $\vec{x}^*$ is the max-min fair allocation.*

PROOF OF LEMMA: We give the proof for the case where $A_{l,i} \in \{0,1\}$; in the general case the proof is similar We proceed by contradiction. Assume that $\vec{x^*}$ is not max-min fair. Then, from Theorem 1.2.3, there is some source i which has no bottleneck. Call L_1 the set (possibly empty) of saturated links used by i, and L_2 the set of other links used by i. For every link $l \in L_1$, we can find some source $\sigma(l)$ which uses l and such that $x^*_{\sigma(l)} > x^*_i$. Define δ by

$$\delta = \frac{1}{5} \min \left\{ \min_{l \in L1}(x^*_{\sigma(l)} - x^*_i),\ \min_{l \in L2}(c_l - (A\vec{x^*})_l) \right\}$$

The term $c_l - (A\vec{x^*})_l$ in the last part of the expression is the unused capacity on link l. We also use the convention that the minimum of an empty set is $+\infty$.

From the convergence of $\vec{x}^{m_n}$ to $\vec{x^*}$, we can find some n_0 such that, for all $n \geq n_0$ and for all j we have

$$x^*_j - \frac{\delta}{I} \leq x^{m_n}_j \leq x^*_j + \frac{\delta}{I} \tag{1.13}$$

where I is the number of sources. Now we construct, for all $n \geq n_0$, an allocation $\vec{y}^n$ which will lead to a contradiction. Define

$$\begin{cases} y^n_i &= x^{m_n}_i + \delta \\ y^n_{\sigma(l)} &= x^{m_n}_{\sigma(l)} - \delta \text{ for } l \in L_1 \\ y^n_j &= x^{m_n}_j \text{ otherwise} \end{cases}$$

We prove first that the allocation $\vec{y}^n$ is feasible. Firstly, we show that the rates are non-negative. From Equation (1.13), we have, for all $l \in L_1$

$$x^{m_n}_{\sigma(l)} \geq x^*_{\sigma(l)} - \frac{\delta}{I} \geq x^*_{\sigma(l)} - \delta$$

thus, from the definition of δ:

$$y^n_{\sigma(l)} \geq x^*_{\sigma(l)} - 2\delta \geq x^*_i + 3\delta \tag{1.14}$$

This shows that $y^n_j \geq 0$ for all j.

Secondly, we show that the total flow on every link is bounded by the capacity of the link. We need to consider only the case of links $l \in (L_1 \cup L_2)$. If $l \in L_1$ then

$$(A\vec{y}^n)_l \leq (A\vec{x}^{m_n})_l + \delta - \delta = (A\vec{x}^{m_n})_l$$

thus the condition is satisfied. Assume now that $l \in L_2$. We have then

$$(A\vec{y}^n)_l = (A\vec{x}^{m_n})_l + \delta$$

Now, from Equation (1.13)

$$(A\vec{x}^{m_n})_l \leq (A\vec{x^*})_l + I\frac{\delta}{I} = (A\vec{x^*})_l + \delta$$

Thus, from the definition of δ:

$$(A\vec{y}^n)_l \leq (A\vec{x^*})_l + 2\delta \leq c$$

which ends the proof that $\vec{y}^n$ is a feasible allocation.

Now we show that, for n large enough, we have a contradiction with the optimality of $\vec{x}^{m_n}$. Consider the expression A defined by

$$A = \sum_j \left(f_{m_n}(y_j^n) - f_{m_n}(x_j^{m_n}) \right)$$

From the optimality of $\vec{x}^{m_n}$, we have: $A \leq 0$.

Now

$$A = f_{m_n}(x_i^{m_n} + \delta) - f_{m_n}(x_i^{m_n}) + \sum_{l \in L_1} \left(f_{m_n}(x_{\sigma(l)}^{m_n} - \delta) - f_{m_n}(x_{\sigma(l)}^{m_n}) \right)$$

From the theorem of intermediate values, there exist numbers c_i^n such that

$$\begin{cases} x_i^{m_n} \leq c_i^n \leq x_i^{m_n} + \delta \\ f_{m_n}(x_i^{m_n} + \delta) - f_{m_n}(x_i^{m_n}) = f'_{m_n}(c_i^n)\delta \end{cases}$$

where f'_{m_n} is the right-derivative of f_{m_n}. Combining with Equation (1.13) we find

$$c_i^n \leq x_i^* + \frac{\delta}{I} + \delta \leq x_i^* + 2\delta \tag{1.15}$$

Similarly, there exist some numbers $c_{\sigma(l)}^n$ such that

$$\begin{cases} x_{\sigma(l)}^{m_n} - \delta \leq c_{\sigma(l)}^n + \delta \leq x_{\sigma(l)}^{m_n} + \delta \\ f_{m_n}(x_{\sigma(l)}^{m_n} - \delta) - f_{m_n}(x_{\sigma(l)}^{m_n}) = -f'_{m_n}(c_{\sigma(l)}^n)\delta \end{cases}$$

and combining with Equation (1.13) we find also

$$c_{\sigma(l)}^n \geq x_i^* + 3\delta \tag{1.16}$$

Thus

$$A = \delta \left(f'_{m_n}(c_i^n) - \sum_{l \in L_1} f'_{m_n}(c_{\sigma(l)}^n) \right)$$

Now f'_{m_n} is wide-sense decreasing (f_{m_n} is concave) thus, combining with Equations (1.15) and (1.16):

$$A \geq \delta \left(f'_{m_n}(x_i^* + 2\delta) - M f'_{m_n}(x_i^* + 3\delta) \right) = \delta f'_{m_n}(x_i^* + 2\delta) \left(1 - M \frac{f'_{m_n}(x_i^* + 3\delta)}{f'_{m_n}(x_i^* + 2\delta)} \right)$$

where M is the cardinal of set L_1. Now from Equation (1.12), the last term in the above equation tends to 1 as n tends to infinity. Now $f'_{m_n} > 0$ from our assumptions thus, for n large enough, we have $A > 0$, which is the required contradiction.

PROOF OF THEOREM: The set of vectors $\vec{x}^m$ is in a compact (= closed + bounded) subset of $\mathbb{R}^I$; thus, it has at least one accumulation point. From the uniqueness of the max-min fair vector, it follows that the set of vectors $\vec{x}^m$ has a unique accumulation point, which is equivalent to saying that

$$\lim_{m \to +\infty} \vec{x}^m = \vec{x}^*$$

□

1.3 DIFFERENT FORMS OF CONGESTION CONTROL

We can consider that there are three families of solutions for congestion control.

Rate Based: Sources know an explicit rate at which they can send. The rate may be given to the source during a negotiation phase; this is the case with ATM or RSVP. In such cases, we have a network with reservation. Alternatively, the rate may be imposed dynamically to the source by the network; this is the case for the ABR class of ATM. In such cases we have a best effort service (since the source cannot be sure of how long a given rate will remain valid), with explicit rate. In the example of the previous section, source 1 would obtain a rate not exceeding 10 kb/s.

Hop by Hop: A source needs some feedback from the next hop in order to send any amount of data. The next hop also must obtain some feedback from the following hop and so on. The feedback may be positive (credits) or negative (backpressure). In the simplest form, the protocol is stop and go. In the example of the previous section, node X would be prevented by node Y from sending source 2 traffic at a rate higher than 10kb/s; source 2 would then be throttled by node X. Hop by hop control is used with full duplex Ethernets using 802.3x frames called "Pause" frames.

End-to-end: A source continuously obtains feedback from all downstream nodes it uses. The feedback is piggybacked in packets returning towards the source, or it may simply be the detection of a missing packet. Sources react to negative feedback by reducing their rate, and to positive feedback by increasing it. The difference with hop-by-hop control is that the intermediate nodes take no action on the feedback; all reactions to feedback are left to the sources. In the example of the previous section, node Y would mark some negative information in the flow of source 2 which would be echoed to the source by destination D2; the source would then react by reducing the rate, until it reaches 10 kb/s, after which there would be no negative feedback. Alternatively, source 2 could detect that a large fraction of packets is lost, reduce its rate, until there is little loss. In broad terms, this is the method invented for Decnet, which is now used after some modification in the Internet.

In the following section we focus on end-to-end control.

1.4 MAX-MIN FAIRNESS WITH FAIR QUEUING

Consider a network implementing fair queuing per flow. This is equivalent to generalized processor sharing (GPS) with equal weights for all flows. With fair queuing, all flows that have data in the node receive an equal amount of service per time slot.

Assume all sources adjust their sending rates such that there is no loss in the network. This can be implemented by using a sliding window, with a window size which is large enough, namely, the window size should be as large as the smallest rate that can be allocated by the source, multiplied by the round trip time. Initially, a source starts sending with a large rate; but in order to sustain the rate, it has to receive acknowledgements. Thus, finally, the rate of the source is limited to the smallest rate allocated by the network nodes. At the node that allocates this minimum rate, the source has a rate which is as high as the rate of any other sources using this node. Following this line of thoughts, the alert reader can convince herself that this node is a bottleneck link for the source, thus the allocation is max-min fair. The detailed proof is complex and is given in [12].

PROPOSITION 1.4.1. *A large sliding window at the sources plus fair queuing in the nodes implements max-min fairness.*

Fixed window plus fair queuing is a possible solution to congestion control. It is implemented in some (old) proprietary networks such as IBM SNA.

Assume now that we relax the assumption that sources are using a window. Assume thus that sources send at their maximum rate, without feedback from the network, but that the network implements fair queuing per flow. Can congestion collapse occur as in Section 1.1.1?

Let us start first with a simple scenario with only one network link of capacity C. Assume there are N sources, and the only constraint on source i is a rate limit r_i. Thus, sources for which $r_i \leq \frac{C}{N}$ have a throughput equal to r_i and thus experience no loss. If some sources have a rate $r_i < \frac{C}{N}$, then there is some extra capacity which can will be used to serve the backlog of other sources. This distribution will follow the algorithm of progressive filling, thus the capacity will be distributed according to max-min fairness, *in the case of a single node*.

In the multiple node case, things are not as nice, as we show now on the following example. The network is as on Figure 1.1. There is 1 source at S1, which sends traffic to D1. There are 10 sources at S2, which all send traffic to D2. Nodes X and Y implement fair queuing per flow. Capacities are :

- 11 Mb/s for link X-Y
- 10 Mb/s for link Y-D1
- 1 Mb/s for link Y-D1

Every source receives 1 Mb/s at node X. The S1- source keeps its share at Y. Every S2 source experiences 90% loss rate at Y and has a final rate of 0.1 Mb/s.

Thus finally, the useful rate for every source is

- 1 Mb/s for a source at S1
- 0.1 Mb/s for a source at S2

The max-min fair share is

- 10 Mb/s for a source at S1
- 0.1 Mb/s for a source at S2

Thus fair queuing alone is not sufficient to obtain max-min fairness. However, we can say that if all nodes in a network implement fair queuing per flow, the throughput for any source s is at least $\min_{l \text{ such that } l \in s} \frac{C_l}{N_l}$, where C_l is the capacity of link l, and N_l is the number of active sources at node l. This implies that congestion collapse as described earlier is avoided.

1.5 ADDITIVE INCREASE, MULTIPLICATIVE DECREASE AND SLOW-START

End-to-end congestion control in packet networks is based on binary feedback and the adaptation mechanisms of additive increase, multiplicative decrease and slow start. We describe here a

motivation for this approach. It comes from the following modeling, from [6]. Unlike the fixed window mentioned earlier, binary feedback does not require fair queuing, but can be implemented with FIFO queues. This is why it is the preferred solution today.

1.5.1 ADDITIVE INCREASE, MULTIPLICATIVE DECREASE

Assume I sources, labeled $i = 1, \ldots, I$ send data at a time dependent rate $x_i(t)$, into a network constituted of one buffered link, of rate c. We assume that time is discrete ($t \in \mathbb{N}$), and that the feedback cycle lasts exactly one time unit. During one time cycle of duration 1 unit of time, the source rates are constant, and the network generates a binary feedback signal $y(t) \in \{0,1\}$, sent to all sources. Sources react to the feedback by increasing the rate if $y(t) = 0$, and decreasing if $y(t) = 1$. The exact method by which this is done is called the adaptation algorithm. We further assume that the feedback is defined by

$$y(t) = [\text{ if } (\sum_{i=1}^{I} x_i(t) \leq c) \text{ then } 0 \text{ else } 1]$$

The value c is the target rate which we wish the system not to exceed. At the same time we wish that the total traffic be as close to c as possible.

We are looking for a linear adaptation algorithm, namely, there must exist constants u_0, u_1 and v_0, v_1 such that

$$x_i(t+1) = u_{y(t)} x_i(t) + v_{y(t)} \tag{1.17}$$

We want the adaptation algorithm to converge towards a fair allocation. In this simple case, there is one single bottleneck and all fairness criteria are equivalent. At equilibrium, we should have $x_i = \frac{c}{I}$. However, a simple adaptation algorithm as described above cannot converge, but in contrast, oscillates around the ideal equilibrium.

We now derive a number of necessary conditions. First, we would like the rates to increase when the feedback is 0, and to decrease otherwise. Call $f(t) = \sum_{i=1}^{I} x_i(t)$. We have

$$f(t+1) = u_{y(t)} f(t) + v_{y(t)} \tag{1.18}$$

Now our condition implies that, for all $f \geq 0$:

$$u_0 f + v_0 > f$$

and

$$u_1 f + v_1 < f$$

This gives the following necessary conditions

$$\begin{cases} u_1 < 1 & \text{and } v_1 \leq 0 \\ \quad \text{or} & \\ u_1 = 1 & \text{and } v_1 < 0 \end{cases} \tag{1.19}$$

and

$$\begin{cases} u_0 > 1 & \text{and } v_0 \geq 0 \\ \quad \text{or} & \\ u_0 = 1 & \text{and } v_0 > 0 \end{cases} \tag{1.20}$$

The conditions above imply that the total rate $f(t)$ remains below c until it exceeds it once, then returns below c. Thus the total rate $f(t)$ oscillates around the desired equilibrium.

Now we also wish to ensure fairness. A first step is to measure how much a given rate allocation deviates from fairness. We follow the spirit of [6] and use as a measure of unfairness the distance between the rate allocation $\vec{x}$ and its nearest fair allocation $\Pi(\vec{x})$, where Π is the orthogonal projection on the set of fair allocations, normalized by the length of the fair allocation (Figure 1.7). In

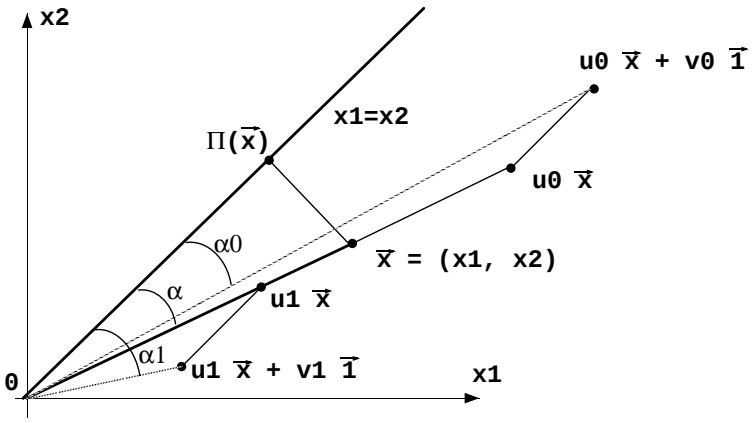


Figure 1.7: The measure of unfairness is $\tan(\alpha)$. The figure shows the effect on fairness of an increase or a decrease. The vector $\vec{1}$ is defined by $\vec{1}_i = 1$ for all i.

other words, the measure of unfairness is

$$d(\vec{x}) = \frac{||\vec{x} - \Pi(\vec{x})||}{||\Pi(\vec{x})||}$$

with $\Pi(\vec{x})_i = \frac{\Sigma_{j=1}^{I} x_j}{I}$ and the norm is the standard euclidian norm defined by $||\vec{y}|| = \sqrt{\Sigma_{j=1}^{I} y_j^2}$ for all $\vec{y}$.

Now we can study the effect of the linear control on fairness. Figure 1.7) illustrates that: (1) when we apply a multiplicative increase or decrease, the unfairness is unchanged; (2) in contrast, an additive increase decreases the unfairness, whereas an additive decrease increases the unfairness.

We wish to design an algorithm such that at every step, unfairness decreases or remains the same, and such that in the long term it decreases. Thus, we must have $v_1 = 0$, in other words, the decrease must be multiplicative, and the increase must have a non-zero additive component. Moreover, if we want to converge as quickly as possible towards, fairness, then we must have $u_0 = 0$. In other words, the increase should be purely additive. In summary we have shown that:

FACT 1.5.1. *Consider a linear adaptation algorithm of the form in Equation 1.17. In order to satisfy efficiency and convergence to fairness, we must have a multiplicative decrease (namely, $u_1 < 1$ and $v_1 = 0$) and a non-zero additive component in the increase (namely, $u_0 \geq 1$ and $v_0 > 0$). If we want to favour a rapid convergence towards fairness, then the increase should be additive only (namely, $u_0 = 1$ and $v_0 > 0$).*

The resulting algorithm is called Additive Increase Multiplicative Decrease (Algorithm 1).

Let us now consider the dynamics of the control system, in the case of additive increase, multiplicative decrease. From Formula 1.18 we see that the *total* amount of traffic oscillates around the

Algorithm 1 Additive Increase Multiplicative Decrease (*AIMD*) with increase term $v_0 > 0$ and decrease factor $0 < u_1 < 1$.

if received feedback is negative **then**
    multiply rate by u_1
else
    add v_0 to rate
end if

optimal value c. In contrast, we see on the numerical examples below that the measure of unfairness converges to 0 (This is always true if the conclusions of Fact (1.5.1) are followed; the proof is left as an exercise to the reader). Figure 1.8 shows some numerical simulations.

The figure also shows a change in the number of active sources. It should be noted that the value of u_1 (the multiplicative decrease factor) plays an important role. A value close to 1 ensures smaller oscillations around the target value; in contrast, a small value of u_1 can react faster to decreases in the available capacity.

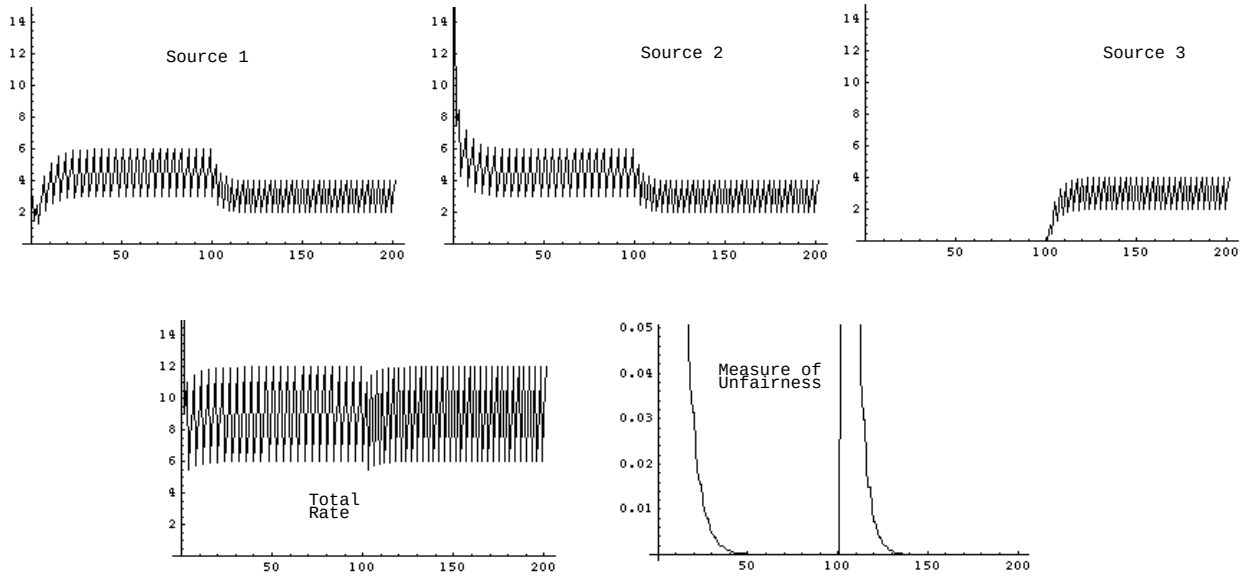
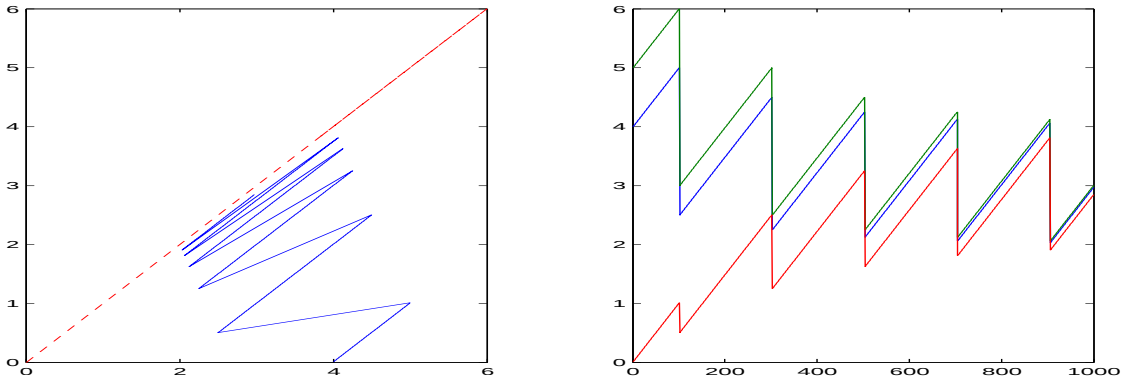


Figure 1.8: Numerical simulations of additive increase, multiplicative decrease. There are three sources, with initial rates of 3 Mb/s, 15 Mb/s and 0. The total link rate is 10 Mb/s. The third source is inactive until time unit 100. Decrease factor = 0.5; increment for additive increase: 1 Mb/s. The figure shows the rates for the three sources, as well as the aggregate rate and the measure of unfairness. The measure of unfairness is counted for two sources until time 100, then for three sources.

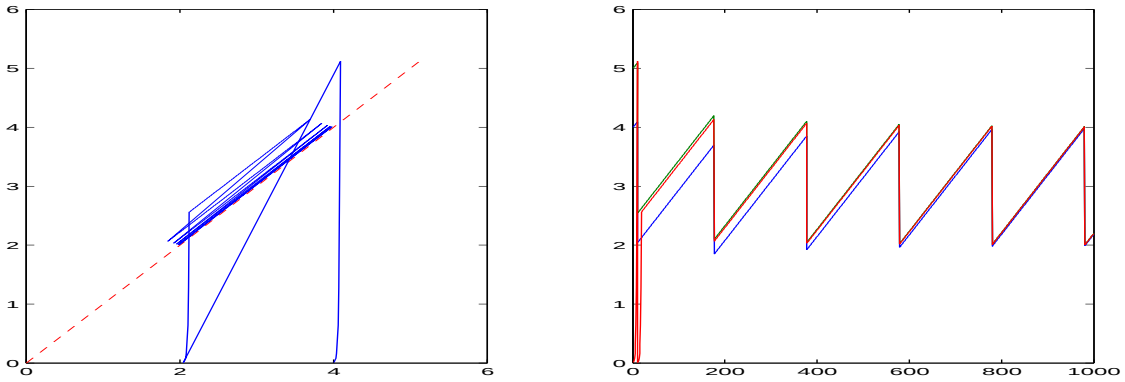
Lastly, we must be aware that the analysis made here ignores the impact of variable and different round trip times.

1.5.2 SLOW START

Slow start is a mechanism that can be combined with Additive Increase Multiplicative Decrease; it applies primarily to the initial state of a flow. It is based on the observation that if we know



(a) Without Slow Start



(b) With Slow Start

Figure 1.9: (a) Three sources applying AIMD, the third starting at time 0 while the other two have reached a high rate. Left: rate of source 1 on x-axis versus rate of source 3. Right: rates of all sources versus time (number of iterations) on x-axis. AIMD parameters: $v_0 = 0.01$; $u_1 = 0.5$. It takes many iterations for the third source to reach the same rate. (b) Same but with the third source applying slow start. It quickly converges to a similar rate. Slow start parameters: $w_0 = 2$, $r_{\max} = 10$

that a flow receives much less than its fair share, then we can deviate from Additive Increase Multiplicative Decrease and give a larger increase to this flow.

We assume that we use Additive Increase Multiplicative Decrease, with the same notation as in the previous subsection, i.e. with additive increase term u_0 and decrease factor v_1.

In Figure 1.8 we showed flows that started with arbitrary rates. In practice, it may not be safe to accept such a behaviour; in contrast, we would like a flow starts with a very small rate. We set this small rate to v_0. Now, a starting flow in the situation, mentioned earlier, of a flow competing with established flows, most probably receives less than all others (Figure 1.9(a)). Therefore, we can, for this flow, try to increase its rate more quickly, for example multiplicatively, until it receives a negative feedback. This is what is implemented in the algorithm called "slow start" (Algorithm 2).

The algorithm maintains both a rate (called x_i in the previous section) and a target rate. Initially, the rate is set to the minimum additive increase v_0 and the target rate to some large, predefined value $r_{\max}$ (lines 1 and next). The rate increases multiplicatively as long as positive feedback is received (line 6). In contrast, if a negative feedback is received, the target rate is decreased multiplicatively (this is applied to the rate achieved so far, line 12) as with AIMD, and the rate is returned to the initial value (lines 13 and next).

The algorithm terminates when and if the rate reached the target rate (line 9). From there on, the

Algorithm 2 Slow Start with the following parameters: AIMD constants $v_0 > 0$, $0 < u_1 < 1$; multiplicative increase factor $w_0 > 1$; maximum rate $r_{\max} > 0$.

```
1: rate ← v_0
2: targetRate ← r_max
3: do forever
4: receive feedback
5: if feedback is positive then
6:     rate ← w_0 · rate
7:     if rate ≥ targetRate then
8:         rate ← targetRate
9:         exit do loop
10:    end if
11: else
12:    targetRate ← max(u_1 · rate, v_0)
13:    rate ← v_0
14: end if
15: end do
```

source applies AIMD, starting from the current value of the rate. See Figure 1.9 for a simulation (Figure 1.10).

Note that what is slow in *slow* start is the starting point v_0, not the increase.

1.6 THE FAIRNESS OF ADDITIVE INCREASE, MULTIPLICATIVE DECREASE WITH FIFO QUEUES

A complete modeling is very complex because it contains both a random feedback (under the form of packet loss) and a random delay (the round trip time, including time for destinations to give feedback). In this section we consider that all round trip times are constant (but may differ from one source to another). The fundamental tool is to produce an ordinary differential equation capturing the dynamics of the network,

1.6.1 A SIMPLIFIED MODEL

Call$x_i(t)$ the sending rate for source i. Call $t_{n,i}$ the nth rate update instant for source i, and let $E_{n,i}$ be the binary feedback: $E_{n,i} = 1$ is a congestion indication; otherwise $E_{n,i} = 0$. Call $1 - \eta_i$ the multiplicative decrease factor and r_i the additive increase component. The source reacts as follows.

$$x(t_{n+1,i}) = E_{n,i}(1-\eta_i)x(t_{n,i}) + (1-E_{n,i})(x(t_{n,i}) + r_i)$$

which we can rewrite as

$$x(t_{n+1,i}) - x(t_{n,i}) = r - E_{n,i}\,(\eta_i x(t_{n,i}) + r_i) \tag{1.21}$$

If some cases, we can approximate this dynamic behaviour by an ordinary differential equation (ODE). The idea, which was developed by Ljung [16] and Kushner and Clark [15], is that the

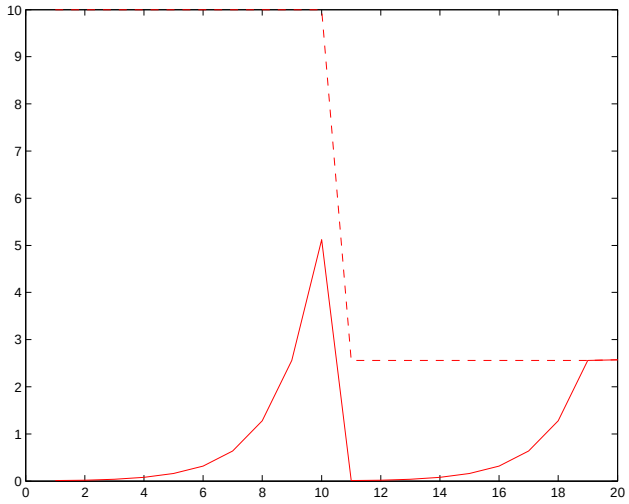


Figure 1.10: Zoom on source 3 of Figure 1.9(b) from times 1 to 20, i.e. during slow start. Dashed line: target rate; plain line: rate. Slow start ends at time 16.

above set of equations is a discrete time, stochastic approximation of a differential equation. The ODE is obtained by writing

$$\begin{aligned} \frac{dx_i}{dt} &= \text{expected rate of change given the state of all sources at time } t \\ &= \text{expected change divided by the expected update interval} \end{aligned} \tag{1.22}$$

The result of the method is that the stochastic system in Equation 1.21 converges, in some sense, towards the global attractor of the ODE in Equation 1.23, under the condition that the ODE indeed has a global attractor [1]. A global attractor of the ODE is a vector $\vec{x}* = (x_i^*)$ towards which the solution converges, under any initial conditions. The convergence is for η_i and r_i tending to 0. Thus the results in this section are only asymptotically true, and must be verified by simulation.

Back to our model, call $\mu_i(t,\vec{x})$ the expectation of $E_{n,i}$, given a set of rates $\vec{x}$; also call $u_i(t)$ the expected update interval for source i. The ODE is:

$$\frac{dx_i}{dt} = \frac{r_i - \mu_i(t,\vec{x}(t))\,(\eta_i x_i(t) + r_i)}{u_i(t)} \tag{1.23}$$

We first consider the original additive increase, multiplicative decrease algorithm, then we will study the special case of TCP.

1.6.2 Additive Increase, Multiplicative Decrease with one Update per RTT

We assume that the update interval is a constant τ_i equal to the round trip time for source i. Thus

$$u_i(t) = \tau_i$$

The expected feedback μ_i is given by

$$\mu_i(t,\vec{x}(t)) = \tau x_i(t) \sum_{l=1}^{L} g_l(f_l(t)) A_{l,i} \tag{1.24}$$

with $f_l(t) = \sum_{j=1}^{I} A_{l,j} x_j(t)$. In the formula, $f_l(t)$ represents the total amount of traffic flow on link l, while $A_{l,i}$ is the fraction of traffic from source i which uses link l. We interprete Equation (1.24) by assuming that $g_l(f)$ is the probability that a packet is marked with a feedback equal to 1 (namely, a negative feedback) by link l, given that the traffic load on link l is expressed by the real number f. Then Equation (1.24) simply gives the expectation of the number of marked packets received during one time cycle by source i. This models accurately the case where the feedback sent by a link is quasi-stationary, as can be achieved by using active queue management such as RED (see later). This also assumes that the propagation time is large compared to the transmission time.

Putting all this together we obtain the ODE:

$$\frac{dx_i}{dt} = \frac{r_i}{\tau_i} - x_i(r_i + \eta_i x_i) \sum_{l=1}^{L} g_l(f_l) A_{l,i} \tag{1.25}$$

with

$$f_l = \sum_{j=1}^{I} A_{l,j} x_j \tag{1.26}$$

In order to study the attractors of this ODE, we identify a Lyapunov for it [20]. To that end, we follow [14] and [10] and note that

$$\sum_{l=1}^{L} g_l(f_l) A_{l,i} = \frac{\partial}{\partial x_i} \sum_{l=1}^{L} G_l(f_l) = \frac{\partial G(\vec{x})}{\partial x_i}$$

where G_l is a primitive of g_l defined for example by

$$G_l(f) = \int_0^f g_l(u) du$$

and

$$G(\vec{x}) = \sum_{l=1}^{L} G_l(f_l)$$

We can then rewrite Equation (1.25) as

$$\frac{dx_i}{dt} = x_i(r_i + \eta_i x_i) \left\{ \frac{r_i}{\tau_i x_i (r_i + \eta_i x_i)} - \frac{\partial G(\vec{x})}{\partial x_i} \right\} \tag{1.27}$$

Consider now the function J_A defined by

$$J_A(\vec{x}) = \sum_{i=1}^{I} \phi(x_i) - G(\vec{x}) \tag{1.28}$$

with

$$\phi(x_i) = \int_0^{x_i} \frac{r_i du}{\tau_i u (r_i + \eta_i u)} = \frac{1}{\tau_i} \log \frac{x_i}{r_i + \eta_i x_i}$$

then we can rewrite Equation (1.27) as

$$\frac{dx_i}{dt} = x_i(r_i + \eta_i x_i) \frac{\partial J_A(\vec{x})}{\partial x_i} \tag{1.29}$$

Now it is easy to see that J_A is strictly concave and therefore has a unique maximum over any bounded region. It follows from this and from Equation (1.29) that J_A is a Lyapunov for the ODE in (1.25), and thus, the ODE in (1.25) has a unique attractor, which is the point where the maximum of J_A is reached. An intuitive explanation of the Lyapunov is as follows. Along any solution $\vec{x}(t)$ of the ODE, we have

$$\frac{d}{dt}J_A(\vec{x}(t)) = \sum_{i=1}^{I} \frac{\partial J_A}{\partial x_i}\frac{dx_i}{dt} = \sum_{i=1}^{I} x_i(r_i+\eta_i x_i)\left(\frac{\partial J_A}{\partial x_i}\right)^2$$

Thus J_A increases along any solution, thus solutions tend to converge towards the unique maximum of J_A.

This shows that the rates $x_i(t)$ converge at equilibrium towards a set of value that maximizes $J_A(\vec{x})$, with J_A defined by

$$J_A(\vec{x}) = \sum_{i=1}^{I} \frac{1}{\tau_i}\log\frac{x_i}{r_i+\eta_i x_i} \; - \; G(\vec{x})$$

INTERPRETATION In order to interpret the previous results, we follow [14] and assume that, calling c_l the capacity of link l, the function g_l can be assumed to be arbitrarily close to δ_{c_l}, in some sense, where

$$\delta_c(f) = 0 \text{ if } f < c \text{ and } \delta_c(f) = +\infty \text{ if } f \geq c$$

Thus, at the limit, the method in [14] finds that the rates are distributed so as to maximize

$$F_A(\vec{x}) = \sum_{i=1}^{I} \frac{1}{\tau_i}\log\frac{x_i}{r_i+\eta_i x_i}$$

subject to the constraints

$$\sum_{j=1}^{I} A_{l,j}x_j \; \leq c_l \text{ for all } l$$

We can learn two things from this. Firstly, the weight given to x_i tends to $-\log\eta_i$ as x_i tends to $+\infty$. Thus, the distribution of rates will tend to favor small rates, and should thus be closer to max-min fairness than to proportional fairness. Secondly, the weight is inversely proportional to the round trip time, thus flows with large round trip times suffer from a negative bias, independent of the number of hops they use.

1.6.3 ADDITIVE INCREASE, MULTIPLICATIVE DECREASE WITH ONE UPDATE PER PACKET

Assume now that a source reacts to every feedback received. Assume that losses are detected immediately, either because the timeout is optimal (equal to the roundtrip time), or because of some other clever heuristic. Then we can use the same analysis as in the previous section, with the following adaptations.

The ODE is still given by Equation (1.23). The expected feedback is now simply equal to the probability of a packet being marked, and is equal to

$$\mu_i(t,\vec{x}) = \sum_{l=1}^{L} g_l(f_l)A_{l,i}$$

The average update interval is now equal to $\frac{1}{x_i}$. Thus the ODE is

$$\frac{dx_i}{dt} = r_i x_i - x_i(r_i + \eta_i x_i) \sum_{l=1}^{L} g_l(f_l) A_{l,i} \tag{1.30}$$

which can also be written as

$$\frac{dx_i}{dt} = x_i(r_i + \eta_i x_i) \left\{ \frac{r_i}{r_i + \eta_i x_i} - \frac{\partial G(\vec{x})}{\partial x_i} \right\} \tag{1.31}$$

Thus a Lyapunov for the ODE is

$$J_B(\vec{x}) = \sum_{i=1}^{I} \frac{r_i}{\eta_i} \log(r_i + \eta_i x_i) \; - \; G(\vec{x})$$

Thus, in the limiting case where the feedback expectation is close to a Dirac function, the rates are distributed so as to maximize

$$F_B(\vec{x}) = \sum_{i=1}^{I} \frac{r_i}{\eta_i} \log(r_i + \eta_i x_i)$$

subject to the constraints

$$\sum_{j=1}^{I} A_{l,j} x_j \; \leq c_l \;\; \text{for all } l$$

The weight given to x_i is close to $\log(\eta_i x_i)$ ($= \log x_i +$ a constant) for large x_i, and is larger than $\log(\eta_i x_i)$ in all cases. Thus, the distribution of rates is derived from proportional fairness, with a positive bias given to small rates. Contrary to the previous case, there is no bias against long round trip times.

In Section 2.1 on page 31 we study the case of TCP.

1.7 SUMMARY

1. In a packet network, sources should limit their sending rate in order to account for the state of the network. Failing to do so might result in congestion collapse.
2. Congestion collapse is defined as a severe decrease in total network throughput when the offered load increases.
3. Maximizing network throughput as a primary objective might lead to large unfairness and is not a viable objective.
4. The objective of congestion control is to provide Pareto efficiency and some form of fairness.
5. Fairness can be defined in various ways: max-min, proportional and variants of proportional.
6. End-to-end congestion control in packet networks is based on the adaptation mechanism of additive increase, multiplicative decrease.
7. Slow start is a mechanism for a source to quickly increase its sending rate (instead of starting upfront at a high rate).

CHAPTER 2

CONGESTION CONTROL FOR BEST EFFORT: INTERNET

In this chapter you learn how the theory of congestion control for best effort is applied in the Internet.

- congestion control algorithms in TCP
- the concept of TCP friendly sources
- random early detection (RED) routers and active queue management

2.1 CONGESTION CONTROL ALGORITHMS OF TCP

We have seen that it is necessary to control the amount of traffic sent by sources, even in a best effort network. In the early days of Internet, a congestion collapse did occur. It was due to a combination of factors, some of them were the absence of traffic control mechanisms, as explained before. In addition, there were other aggravating factors which led to "avalanche" effects.

- IP fragmentation: if IP datagrams are fragmented into several packets, the loss of one single packet causes the destination to declare the loss of the entire datagram, which will be retransmsitted. This is addressed in TCP by trying to avoid fragmentation. With IPv6, fragmentation is possible only at the source and for UDP only.
- Go Back n at full window size: if a TCP sender has a large offered window, then the loss of segment n causes the retransmission of all segments starting from n. Assume only segment n was lost, and segments $n+1,\ldots,n+k$ are stored at the receiver; when the receiver gets those segments, it will send an ack for all segments up to $n+k$. However, if the window is large, the ack will reach the sender too late for preventing the retransmissions. This has been addressed in current versions of TCP where all timers are reset when one expires.
- In general, congestion translates into larger delays as well (because of queue buildup). If nothing is done, retransmission timers may become too short and cause retransmissions of

data that was not yet acked, but was not yet lost. This has been addressed in TCP by the round trip estimation algorithm.

Congestion control has been designed right from the beginning in wide-area, public networks (most of them are connection oriented using X.25 or Frame Relay), or in large corporate networks such as IBM's SNA. It came as an afterthought in the Internet. In connection oriented network, congestion control is either hop-by-hop or rate based: credit or backpressure per connection (ATM LANs), hop-by-hop sliding window per connection (X.25, SNA); rate control per connection (ATM). They can also use end-to-end control, based on marking packets that have experienced congestion (Frame Relay, ATM). Connectionless wide area networks all rely on end-to-end control.

In the Internet, the principles are the following.

- TCP is used to control traffic
- the rate of a TCP connection is controlled by adjusting the window size
- additive increase, multiplicative decrease and slow start, as defined in Chapter 1, are used
- the feedback from the network to sources is packet loss. It is thus assumed that packet loss for reasons other than packet dropping in queues is negligible. In particular, all links should have a negligible error rate.

One implication of these design decisions is that only TCP traffic is controlled. The reason is that, originally, UDP was used only for short transactions. Applications that do not use TCP have to either be limited to LANs, where congestion is rare (example: NFS) or have to implement in the application layer appropriate congestion control mechanisms (examplee.g., QUIC or some audio and video applications). We will see later what is done in such situations.

Only long lasting flows are the object of congestion control. There is no congestion control mechanism for short lived flows.

The detailed mechanisms are described below. Over the years, many variants of TCP congestion control emerged. We describe here only a few representative ones. The concepts are heavily influenced by the historical version, TCP RENO and its variant TCP NEW RENO, which we describe next. Then we also describe TCP CUBIC and Data Center TCP, two widespread variants.

2.1.1 CONGESTION WINDOW

Remember that, with the sliding window protocol concept (used by TCP), the window size W (in bits or bytes) is equal to the maximum number of unacknowledged data that a source may send. Consider a system where the source has infinite data to send; assume the source uses a FIFO queue as send buffer, of size W. At the beginning of the connection, the source immediately fills the buffer which is then dequeued at the rate permitted by the line. Then the buffer can receive new data as old data is acknowledged. Let T be the average time you need to wait for an acknowledgement to come back, counting from the instant the data is put into the FIFO queue. This system is an approximate model of a TCP connection for a source which is infinitely fast and has an infinite amount of data to send. By Little's formula applied to the FIFO queue, the throughput θ of the TCP connection is given by (prove this as an exercize):

$$\theta = \frac{W}{T} \tag{2.1}$$

The delay T is equal to the propagation and transmission times of data and acknowledgement, plus the processing time, plus possible delays in sending acknowledgement. If T is fixed, then controlling W is equivalent to controlling the connection rate θ. This is the method used in the Internet. However, in general, T depends also on the congestion status of the networks, through queueing delays. Thus, in periods of congestions, there is a first, automatic congestion control effect: sources reduce their rates whenever the network delay increases, simply because the time to get acknowledgements increase. This is however a side effect, which is not essential in the TCP congestion control mechanism.

TCP defines a variable called congestion window (`cwnd`); the window size W is then given by

$$W = \min(\texttt{cwnd}, \texttt{offeredWindow})$$

Remember that `offeredWindow` is the window size advertized by the destination. In contrast, `cwnd` is computed by the source.

The value of `cwnd` is decreased when a loss is detected, and increased otherwise. In the rest of this section we describe the details of the operation.

A TCP connection is, from a congestion control point of view, in one of three phases.

- slow start: after a loss detected by timeout
- fast recovery: after a loss detected by fast retransmit
- congestion avoidance: in all other cases.

The variable `cwnd` is updated at phase transitions, and when useful acknowledgements are received. Useful acknowledgements are those which increase the lower edge of the sending window, i.e. are not duplicate.

2.1.2 SLOW START AND CONGESTION AVOIDANCE

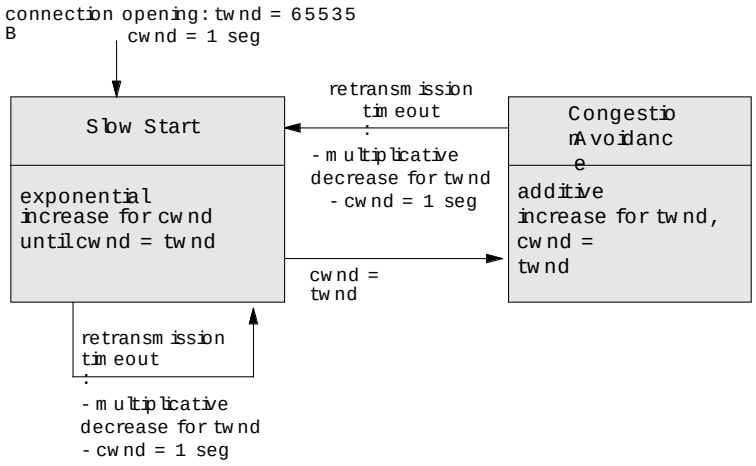


Figure 2.1: Slow Start and Congestion Avoidance, showing the actions taken in the phases and at phase transitions.

In order to simplify the description, we first describe an incomplete system with only two phases: slow start and congestion avoidance. This corresponds to a historical implementation (TCP Tahoe)

where losses are detected by timeout only (and not by the fast retransmit heuristic). According to the additive increase, multiplicative decrease principle, the window size is divided by 2 for every packet loss detected by timeout. In contrast, for every useful acknowledgement, it is increased according to a method described below, which produces a roughly additive increase. At the beginning of a connection, slow start is used (Algorithm 2 on page 25). Slow start is also used after every loss detected by timeout. Here the reason is that we expect most losses to be detected instead by fast retransmit, and so losses detected by timeout probably indicate a severe congestion; in this case it is safer to test the water carefully, as slow start does.

Similar to "targetRate" in Algorithm 2, a supplementary variable is used, which we call the target window (`twnd`) (it is called `ssthresh` in RFC 5681). At the beginning of the connection, or after a timeout, `cwnd` is set to 1 segment and a rapid increase based on acknowledgements follows, until `cwnd` reaches `twnd`.

The algorithm for computing `cwnd` is shown on Figure 2.1. At connection opening, `twnd` has the maximum value (64KB by default, more if the window scale option is used – this corresponds to the $r_{\max}$ parameter of Algorithm 2), and `cwnd` is one segment. During slow start, `cwnd` increases exponentially (with increase factor $w_0 = 2$). Slow start ends whenever there is a packet loss detected by timeout or `cwnd` reaches `twnd`. During congestion avoidance, `cwnd` increases according to the additive increase explained later. When a packet loss is detected by timeout, `twnd` is divided by 2 and slow start is entered or re-entered, with `cwnd` set to 1.

Note that `twnd` and `cwnd` are equal in the congestion avoidance phase.

Figure 2.2 shows an example built with data from [4]. Initially, the connection congestion state is slow-start. Then `cwnd` increases from one segment size to about 35 KB, (time 0.5) at which point the connection waits for a missing non duplicate acknowledgement (one packet is lost). Then, at approximately time 2, a timeout occurs, causing `twnd` to be set to half the current window, and `cwnd` to be reset to 1 segment size. Immediately after, another timeout occurs, causing another reduction of `twnd` to 2 $\times$ `segment size` and of `cwnd` to 1 segment size. Then, the slow start phase ends at point A, as one acknowledgement received causes `cwnd` to equal `twnd`. Between A and B, the TCP connection is in the congestion avoidance state. `cwnd` and `twnd` are equal and both increase slowly until a timeout occurs (point B), causing a return to slow start until point C. The same pattern repeats later. Note that some implementations do one more multiplicative increase when `cwnd` has reached the value of `twnd`.

The slow start and congestion avoidance phases use three algorithms for decrease and increase, as shown on Figure 2.1.

1. Multiplicative Decrease for `twnd`

```
twnd = 0.5 * current window size
twnd = max (twnd, 2 * segment size)
```

2. Additive Increase for `twnd`

```
for every useful acknowledgement received :
        twnd = twnd + (segment size) * (segment size) / twnd
        twnd = min (twnd, maximum window size)
```

3. Exponential Increase for `cwnd`

```
for every useful acknowledgement received :
        cwnd = cwnd + (segment size)
```

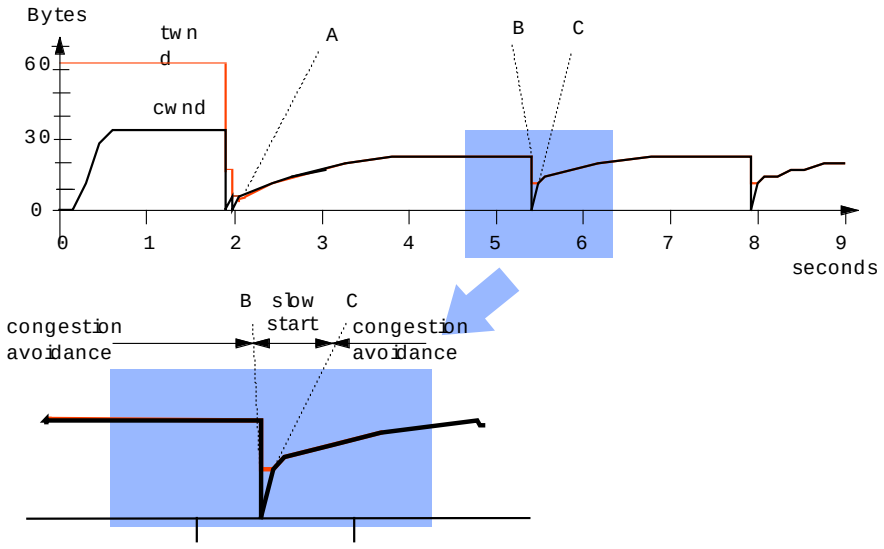


Figure 2.2: Typical behaviour with slow start and congestion avoidance, constructed with data from [4]. It shows the values of `twnd` and `cwnd`.

```
if (cwnd == twnd) then move to congestion avoidance
```

In order to understand the effect of the Additive Increase algorithm, remember that TCP windows are counted in bytes, not in packets. Assume that `twnd` $= w$ `segment size`, thus w is the size counted in packets, assuming all packets have a size equal to `segment size`. Thus, for every acknowledgement received, `twnd/segment size` is increased by $1/w$, and it takes a full window to increment w by one. This is equivalent to an additive increase if the time to receive the acknowledgments for a full window is constant. Figure 2.3 shows an example.

Note that additive increase is applied during congestion avoidance, during which phase we have `twnd` $=$ `cwnd`.

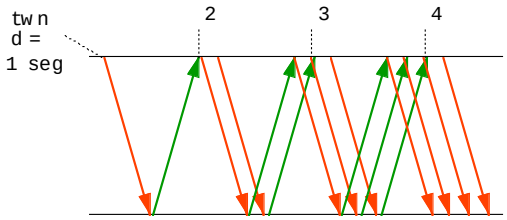


Figure 2.3: The additive increase algorithm.

The Exponential Increase algorithm is applied during the slow start phase. The effect is to increase the window size until `twnd`, as long as acknowledgements are received. Figure 2.4 shows an example.

Finally, Figure 2.5 illustrates the additive increase, multiplicative decrease principle and the role of slow start. Do not misinterpret the term “slow start”: it is in reality a phase of rapid increase; what is slow is the fact that `cwd` is set to 1. The slow increase is during congestion avoidance, not during slow start.

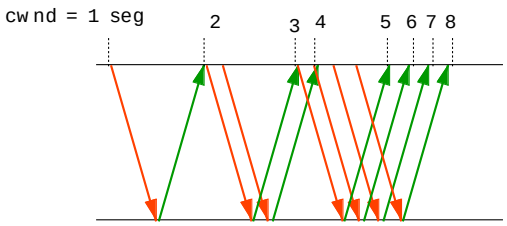


Figure 2.4: The exponential increase algorithm for `cwnd`.

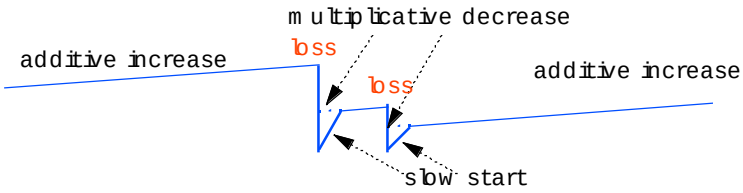


Figure 2.5: Additive increase, Multiplicative decrease and slow start.

2.1.3 FAST RECOVERY

As mentioned earlier, the full specification for TCP involves a third state, called Fast Recovery, which we describe now. Remember from the previous section that when a loss is detected by timeout, the target congestion window size `twnd` is divided by 2 (Multiplicative Decrease for `twnd`) but we also go into the slow start phase in order to avoid bursts of retransmissions.

However, this is not very efficient if an isolated loss occurs. Indeed, the penalty imposed by slow start is large; it will take about $\log n$ round trips to reach the target window size `twnd`, where $n = \texttt{twnd/segment size}$. This is in particular to severe if the loss is isolated, corresponding to a mild negative feedback. Now with the current TCP, isolated losses are assumed to be detected and repaired with the Fast Retransmit procedure.

Therefore, we add a different mechanism for every loss detected by Fast Retransmit. The procedure is as follows.

- when a loss is detected by Fast Retransmit (triplicate ack), then run Multiplicative Decrease for `twnd` as described in the previous section.
- Then enter a temporary phase, called Fast Recovery, until the loss is repaired. When entering this phase, temporarily keep the congestion window high in order to keep sending. Indeed, since an ack is missing, the sender is likely to be blocked, which is not the desired effect:

```
cwnd = twnd + 3 *seg /* exponential increase */
cwnd = min(cwnd, 65535)
retransmit missing segment (say n)
```

- Then continue to interprete every received ack as a positive signal, at least until the lost is repaired, running the exponential increase mechanism:

```
duplicate ack received ->
   cwnd = cwnd + seg              /* exponential increase */
   cwnd = min(cwnd, 65535)
```

```
        send following segments if window allows

    ack for segment n received ->
       go into to cong. avoidance
```

Figure 2.6 shows an example.

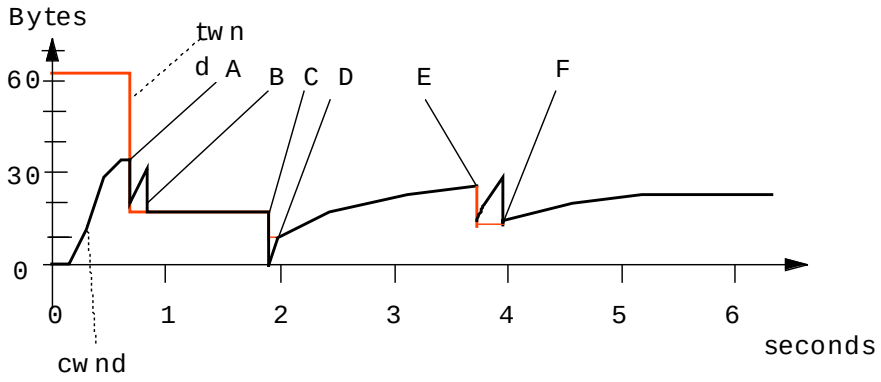


Figure 2.6: A typical example with slow start (C-D) and fast recovery (A-B and E-F), constructed with data from [4]. It shows the values of `twnd` and `cwnd`.

If we combine the three phases described in this and the previous section, we obtain the complete diagram, shown on Figure 2.7.

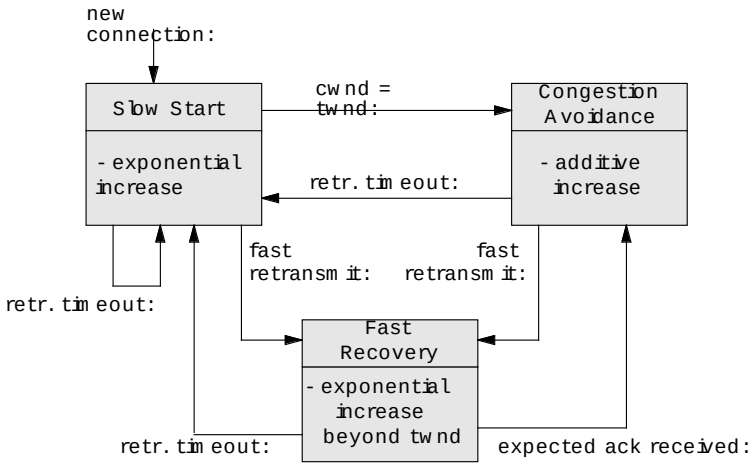


Figure 2.7: Slow Start, Congestion Avoidance and Fast Retransmit showing the phase transitions.

2.1.4 SUMMARY AND COMMENTS

In summary for that section, we can say that the congestion avoidance principle for the Internet, used in TCP, is additive increase, multiplicative decrease. The sending rate of a source is governed

by a target window size `twnd`. The principle of additive increase, multiplicative decrease is summarized as follows. At this point you should be able to understand this summary; if this is not the case, take the time to read back the previous sections. See also Figure 2.5 for a synthetic view of the different phases.

- when a loss is detected (timeout or fast retransmit), then `twnd` is divided by 2 ("Multiplicative Decrease for `twnd`")
- In general (namely in the congestion avoidance phase), for every useful (i.e. non duplicate) ack received, `twnd` is increased linearly ("Additive Increase for `twnd`")
- Just after a loss is detected a special transient phase is entered. If the loss is detected by timeout, this phase is slow start; if the loss is detected by fast retransmit, the phase is fast recovery. At the beginning of a connection, the slow start phase is also entered.
  During such a transient phase, the congestion window size is different from `twnd`. When the transient phase terminates, the connection goes into the congestion avoidance phase. During congestion avoidance, the congestion window size is equal to `twnd`.

2.2 ANALYSIS OF TCP RENO

2.2.1 THE FAIRNESS OF TCP RENO

In this section we determine the fairness of TCP, assuming all round trip times remain constant over time. We can apply the analysis in Section 1.6 and use the method of the ODE.

TCP differs slightly from the plain additive increase, multiplicative decrease algorithm. Firstly, it uses a window rather than a rate control. We can approximate the rate of a TCP connection, if we assume that transients due to slow start and fast recovery can be neglected, by $x = \frac{w}{\tau}$, where w is equal to `cwnd` and τ is the round trip time, assumed to be constant. Secondly, the increase in rate is not strictly additive; in contrast, the window is increased by $\frac{1}{w}$ for every positive acknowledgement received. The increase in rate at every positive acknowledgement is thus equal to $\frac{K}{w\tau} = \frac{K}{x\tau^2} =$ where K is a constant. If the unit of data is the packet, then $K = 1$; if the unit of data is the bit, then $K = L^2$, where L is the packet length in bits.

In the sequel we consider some variation of TCP where the window increase is still given by $\frac{K}{w\tau}$, but where K is not necessarily equal to $(1 \text{ packet})^2$ and need not be the same for all TCP connections.

We use the same notation as in Section 1.6 and call $x_i(t)$ the rate of source i. The coefficient K may now depend on the connection i and is denoted with K_i. The ODE is obtained by substituting r_i by $\frac{K_i}{x_i\tau_i^2}$ in Equation (1.30) on page 29:

$$\frac{dx_i}{dt} = \frac{K_i}{\tau_i^2} - (\frac{K_i}{\tau_i^2} + \eta_i x_i^2) \sum_{l=1}^{L} g_l(f_l) A_{l,i} \tag{2.2}$$

which can also be written as

$$\frac{dx_i}{dt} = (\frac{K_i}{\tau_i^2} + \eta_i x_i^2) \left\{ \frac{\frac{K_i}{\tau_i^2 \eta_i}}{\frac{K_i}{\tau_i^2 \eta_i} + x_i^2} - \frac{\partial G(\vec{x})}{\partial x_i} \right\} \tag{2.3}$$

This shows that the rates $x_i(t)$ converge at equilibrium towards a set of value that maximizes $J_C(\vec{x})$, with J_C defined by

$$J_C(\vec{x}) = \sum_{i=1}^{I} \frac{1}{\tau_i}\sqrt{\frac{K_i}{\eta_i}} \arctan \frac{x_i\tau_i}{\sqrt{\frac{K_i}{\eta_i}}} \; - \; G(\vec{x}) \tag{2.4}$$

Thus, in the limiting case where the feedback expectation is close to a Dirac function, the rates are distributed so as to maximize

$$F_C(\vec{x}) = \sum_{i=1}^{I} \frac{1}{\tau_i}\sqrt{\frac{K_i}{\eta_i}} \arctan \frac{x_i\tau_i}{\sqrt{\frac{K_i}{\eta_i}}}$$

subject to the constraints

$$\sum_{j=1}^{I} A_{l,j}x_j \; \leq c_l \;\; \text{for all } l$$

THE BIAS OF TCP RENO AGAINST LONG ROUND TRIP TIMES If we use the previous analysis with the standard parameters used with TCP-Reno, we have $\eta_i = 0.5$ for all sources and (the unit of data is the packet)

$$K_i = 1$$

With these values, the expected rate of change (right hand side in Equation (2.2)) is a decreasing function of τ_i. Thus, the adaptation algorithm of TCP contains a negative bias against long round trip times. More precisely, the weight given to source i is

$$\frac{\sqrt{2}}{\tau_i} \arctan \frac{x_i\tau_i}{\sqrt{2}}$$

If x_i is very small, then this is approximately equal to $2x_i$, independent of τ_i. For a very large x_i, it is approximately equal to $\frac{\sqrt{2}\Pi}{2\tau_i}$. Thus, the negative bias against very large round trip times is important only in the cases where the rate allocation results into a large rate.

Note that the bias against long round trip times comes beyond and above the fact that, like with proportional fairness, the adaptation algorithm gives less to sources using many resources. Consider for example two sources with the same path, except for an access link which has a very large propagation delay for the second source. TCP will give less throughput to the second one, though both are using the same set of resources.

We can correct the bias of TCP against long round trip times by changing K_i. Remember that K_i is such that the window w_i is increased for every positive acknowledgement by $\frac{K_i}{w_i}$. Equation (2.2) shows that we should let K_i be proportional to τ_i^2. This is the modification proposed in [9] and [13]. Within the limit of our analysis, this would eliminate the non-desired bias against long round trip times. Note that, even with this form of fairness, connections using many hops are likely to receive less throughput; but this is not because of a long round trip time.

If we compare the fairness of TCP to proportional fairness, we see that the weight given to x_i is bounded both as x_i tends to 0 or to $+\infty$. Thus, it gives more to smaller rates than proportional fairness.

In summary, we have proven that:

PROPOSITION 2.2.1. *TCP tends to distribute rates so as to maximize the utility function J_C defined in Equation (2.4).*

- *If the window increase parameter is as with TCP Reno ($K_i = 1$ for all sources), then TCP has a non-desired negative bias against long round trip times.*
- *If in contrast the bias is corrected (namely, the window w_i is increased for every positive acknowledgement by $\frac{K\tau_i^2}{w_i}$), then the fairness of TCP is a variant of proportional fairness which gives more to smaller rates.*

Thus, TCP Reno distributes rates equally among connections having the same paths, but has a bias against connections

1. with a large number of hops
2. or with a large round trip time

The first bias is a result of proportional fairness and can be considered to be justified; the second bias is less justified and is indeed a problem for connections using links with large latency, eg., satellite links.

2.2.2 THE LOSS-THROUGHPUT FORMULA

A coarser, but more enlightening, analysis of the performance of TCP Reno leads to a simple relation between the loss probability experienced by a TCP connection and its throughput, assuming the application has an infinite amount of data to send (i.e., the bottleneck is the network). We expect the throughput of TCP to decrease with the loss probability; this is indeed captured by the following result.

THEOREM 2.2.1 (TCP loss - throughput formula [19]). *Consider a TCP Reno connection with constant round trip time τ and constant packet size L; assume that the network is stationary, that the transmission time is negligible compared to the round trip time, that losses are rare and that the time spent in slow start or fast recovery is negligible; then the average throughput θ (in bits/s) and the average packet loss ratio q are linked by the relation*

$$\theta \approx \frac{L}{\tau}\frac{C}{\sqrt{q}} \tag{2.5}$$

with $C = 1.22$

PROOF: We consider that we can neglect the phases where `twnd` is different from `cwnd` and we can thus consider that the connection follows the additive increase, multiplicative decrease principle. We assume that the network is stationary; thus the connection window size `cwnd` oscillates as illustrated on Figure 2.8. The oscillation is made of cycles of duration T_0. During one cycle, `cwnd` grows until a maximum value W, then a loss is encountered, which reduces `cwnd` to $\frac{W}{2}$. Now from our assumptions, exactly a full window is sent per round trip time, thus the window increases by one packet per round trip time, from where it follows that

$$T_0 = \frac{W}{2}\tau$$

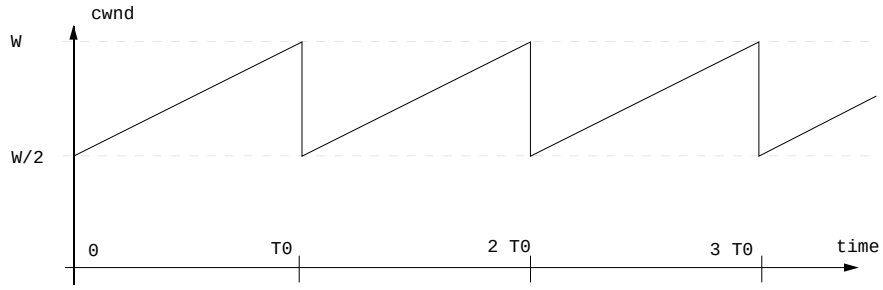


Figure 2.8: The evolution of `cwnd` under the assumptions in the proposition.

The sending rate is approximately $\frac{W(t)}{\tau}$. It follows also that the number of packets sent in one cycle, N, is given by

$$N = \int_0^{T_0} \frac{W(t)}{\tau} dt = \frac{3}{8} W^2$$

Now one packet is lost per cycle, so the average loss ratio is

$$q = 1/N$$

We can extract W from the above equations and obtain

$$W = 2\sqrt{\frac{2}{3}} \frac{1}{\sqrt{q}}$$

Now the average throughput is given by

$$\theta = \frac{(N-1)}{T_0} L = \frac{\left(\frac{1}{q} - 1\right) L}{\sqrt{\frac{2}{3}} \frac{1}{\sqrt{q}} \tau}$$

If q is very small this can be approximated by

$$\theta \approx \frac{L}{\tau} \frac{C}{\sqrt{q}}$$

with $C = \sqrt{\frac{3}{2}}$. □

Formula 2.5 is taken as a basis for designing alternatives to TCP Reno. It can be shown that the formula still holds, though with a slightly different constant, if more realistic modelling assumptions are taken [7, 17].

2.3 OTHER MECHANISMS FOR TCP CONGESTION CONTROL

2.3.1 CUBIC

WHY CUBIC. The motivation of this variant is networks where the bandwidth-delay product is very large, i.e., when both the round trip time and the bit-rate available to one connection are large

("Long Fat Networks"). Consider the typical saw-tooth behaviour of TCP (and AIMD) illustrated in Figure 2.8. The available bit rate is $b = WL/\tau$ where τ is the round-trip time and L is the packet length; also recall that the duration T_0 of one oscillation is $T_0 = W/2\tau$. Thus

$$T_0 = \frac{b\tau^2}{2L}$$

For example, if the round-trip time is $\tau = 100$ msec, the available bit rate is 10Mb/s and the packet size is 1250 bytes, then the time T_0 taken by one oscillation is equal to 5 sec, which is small. But if the available bit rate becomes 10Gb/s, T_0 becomes 1 h 23 mn ! In the latter case, the additive increase is too slow and chances are that the connection is over before completing even a single oscillation. TCP CUBIC attempts to be a replacement for TCP Reno that alleviates this problem in long fat networks but behaves the same as TCP otherwise [11, 24].

MECHANISMS OF CUBIC. CUBIC keeps multiplicative decrease and slow start as with TCP Reno, and keeps the same rules for exiting congestion avoidance (on a loss event) as we have seen in Section 2.1. However, when a loss is detected by duplicate acknowledgements, the multiplicative decrease factor is 0.7 instead of 0.5 (i.e., when a loss is detected by duplicate acknowledgements, the congestion window is set to $0.7W_{\max}$ where $W_{\max}$ is the value of the congestion window just before the loss event). The smaller window reduction intends to reduce the amplitude of the oscillations.

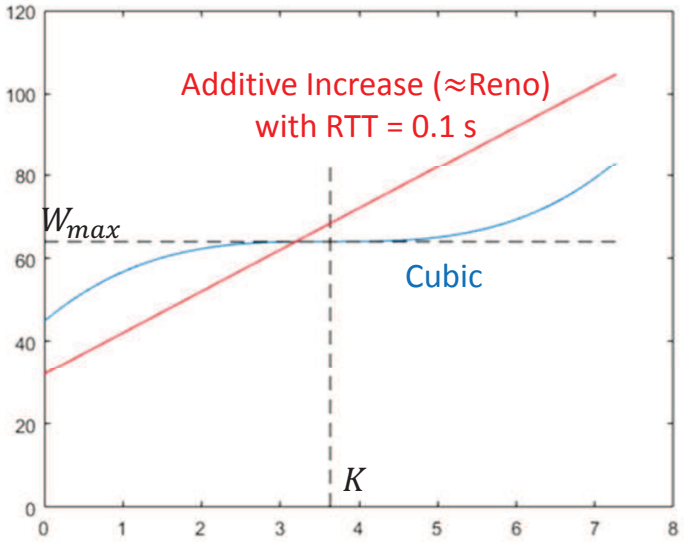


Figure 2.9: The window increase with TCP CUBIC.

But the major difference introduced by CUBIC is the replacement of additive increase during congestion avoidance phase. The linear growth during additive increase is replaced by a cubic function, as illustrated in Figure 2.9. Specifically, CUBIC uses a function $W(t)$ to compute the congestion window at time t seconds after a loss event, given by

$$W(t) = W_{\max} + a(t-K)^3 \tag{2.6}$$

where a is a constant and K is computed such that $W(0) = 0.7W_{\max}$. As illustrated in the figure, the growth of the congestion window is concave (slower than linear) as long as $W_{\max}$ is not attained, and convex (faster than linear) after that point. Thus, CUBIC increases slowly until the value $W_{\max}$ is reached; at this value, the increase is the slowest. This is indeed a very safe behaviour, since in stable network conditions, $W_{\max}$ is the region where congestion is likely to start, If, in contrast, the network has plenty of capacity when the congestion window reaches $W_{\max}$, then the window increase past this point is convex, i.e. increases fast. This is what allows CUBIC to accelerate the window growth in long fat networks.

However, for small values of $W_{\max}$ and round-trip time, which occur in non long fat networks, it can happen that $W(t)$ is smaller than $W_{\text{AIMD}}(t)$, the value of the congestion window that would be obtained with additive increase, multiplicative increase (Figure 2.10). Since CUBIC aims to obtain as much throughput as TCP Reno, CUBIC sets the value of the congestion window during the congestion avoidance phase to

$$W_{\text{CUBIC}}(t) = \max\{W(t), W_{\text{AIMD}}(t)\} \tag{2.7}$$

There is a small issue here. The parameters of AIMD should be adapted in order to account for the multiplicative decrease factor of 0.7 instead of 0.5. Indeed, TCP Reno increases the window size by 1 packet per round trip time in congestion avoidance and multiplies the window by 0.5 at a loss event. Therefore, $W_{\text{AIMD}}(t)$ is not equal to the value of the congestion window with TCP Reno, but to the value obtained by a hypothetical AIMD that multiplies the window by 0.7 at a loss event. In order to have a similar behaviour as TCP Reno, this version of AIMD should increases the window size by $r < 1$ packet per round-trip time during congestion avoidance, in order to compensate for the smaller decrease. Specifically, we can determine r by requiring that the loss throughput formula gives the same value for both cases. The loss throughput formula in Equation (2.5) was derived for the case of TCP Reno; it can easily be modified to the case where the additive increase is r packets per round-trip time and the decrease factor is β. We find, in this case, that Equation (2.5) should be modified such that C is replaced by

$$C_{r,\beta} = \sqrt{\frac{r(1+\beta)}{2(1-\beta)}} \tag{2.8}$$

The case of TCP Reno corresponds to $r = 1, \beta = 0.5$ and gives $C_{1,0.5} = \sqrt{\frac{3}{2}}$ as expected. For CUBIC, $\beta = 0.7$ and the value of r should be such that $C_{r,0.7} = C_{1,0.5}$ which gives $r = 3\frac{1-\beta}{1+\beta} = 0.529$. In summary, CUBIC uses Equation (2.7) with $W_{\text{AIMD}}(t) = 0.529\frac{t}{RTT}$. When CUBIC uses $W_{\text{AIMD}}(t)$, i.e. when $W_{\text{AIMD}}(t) > W(t)$, we say that CUBIC is in the "TCP-friendly" region.

There is an additional mechanism called "Fast Convergence", which is used to decrease the congestion window size at a loss event more severely when $W_{\max}$ is decreasing from one congestion avoidance phase to the next, see [24] for details.

CUBIC's increase of the congestion window is independent of the round-trip time, therefore we might expect it to remove the undesired bias of TCP Reno against large round trip time. However, as we see next, CUBIC does remove some of the bias against large round trip times, but not entirely. This is because the sending rate is proportional to the window and the inverse of the round-trip time: the increase in *rate* is slower for large round trip times.

ANALYSIS OF CUBIC. We can extend the loss throughput formula of TCP Reno in Theorem 2.2.1 to the cubic increase function, using the same method.

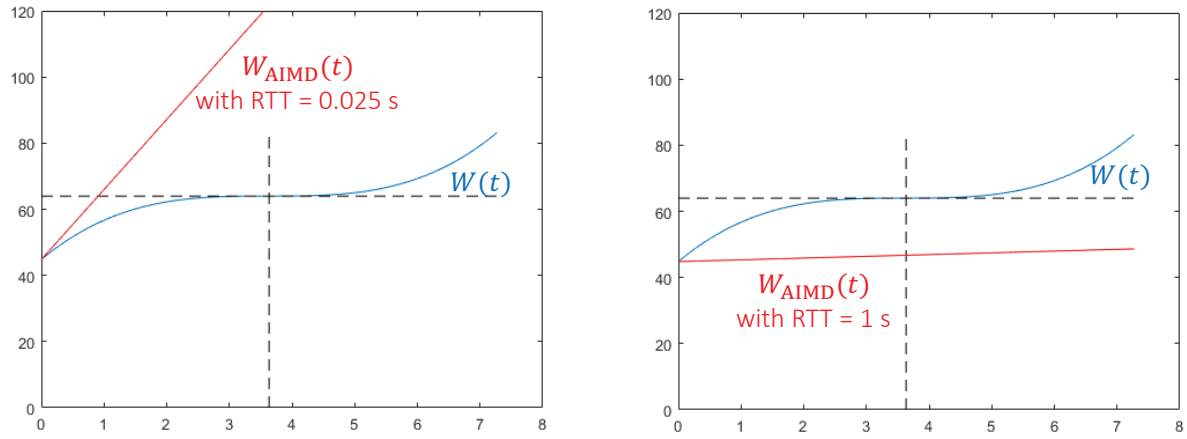


Figure 2.10: The window increase with TCP CUBIC. Left: RTT is small, the cubic window increase is less than the additive increase and CUBIC uses $W_{\text{AIMD}}(t)$. Right: the converse holds when RTT is large, and CUBIC uses $W(t)$.

THEOREM 2.3.1 (Loss - throughput formula with cubic increase). *Consider a TCP connection with constant round trip time τ and constant packet size L; assume that the network is stationary, that the transmission time is negligible compared to the round trip time, that losses are rare and that the time spent in slow start or fast recovery is negligible; also assume that when a loss event occurs, the window is multiplied by a factor βW. Last, assume that the increase function is a cubic function, given by*

$$W(t) = W_{\max} + a(t-K)^3 \tag{2.9}$$

where $W_{\max}$ is the window size just before the loss event that triggered a new congestion avoidance phase, a is a constant and K is computed such that $W(0) = \beta W_{\max}$.

Then the average throughput θ (in bits/s) and the average packet loss ratio q are linked by the relation

$$\theta \approx \frac{L}{\tau^{0.25}} \frac{C_{cubic}}{q^{0.75}} \tag{2.10}$$

with $C_{cubic} = \left(a\frac{3+\beta}{4(1-\beta)}\right)^{0.25}$.

PROOF: The proof is similar to the proof of Theorem 2.2.1. First observe that, with the conditions in the theorem, the window increase is always in the concave phase and the window size is periodic with period K. Let W be the max window size. The number of packets sent in one period is

$$N = \frac{1}{\tau}\int_0^K \left(W + a(t-K)^3\right) dt = \frac{1}{\tau}\left(WK - a\frac{K^4}{4}\right)$$

The function $W(t)$ satisfies $W(0) = \beta W$ hence, after some algebra:

$$W = \frac{a}{1-\beta}K^3$$

Combining with the previous equation gives

$$N = \frac{1}{\tau}\alpha K^4 \text{ with } \alpha = a\frac{3+\beta}{4(1-\beta)}$$

There is one loss event per period of duration K hence the loss probability is $q = 1/N$. Thus

$$K = \frac{\tau^{\frac{1}{4}}}{(\alpha q)^{\frac{1}{4}}}$$

$N-1$ packets are successfully sent every K seconds hence the throughput is

$$\theta = L\frac{N-1}{K} \approx L\frac{N}{K} = \frac{L}{q} \times \frac{(\alpha q)^{\frac{1}{4}}}{\tau^{\frac{1}{4}}} = \frac{L\alpha^{\frac{1}{4}}}{\tau^{\frac{1}{4}} q^{\frac{3}{4}}}$$

□

This theorem can be used to obtain a very coarse loss-throughput formula for CUBIC. First, using the values $a = 0.4$ and $\beta = 0.7$ we find $C_{\text{cubic}} = 1.054$. Second, observe that CUBIC's window increase does not use the cubic function $W(t)$ but is linear when CUBIC operates in the TCP-friendly region; in particular, CUBIC's throughput is at least that of TCP Reno, which, according to Equation (2.5), is $\frac{L}{\tau}\frac{1.22}{\sqrt{q}}$. An approximate formula is thus

$$\theta_{\text{CUBIC}} \approx \max\left\{\frac{L}{\tau}\frac{1.22}{\sqrt{q}}, \frac{L}{\tau^{0.25}}\frac{1.054}{q^{0.75}}\right\}$$

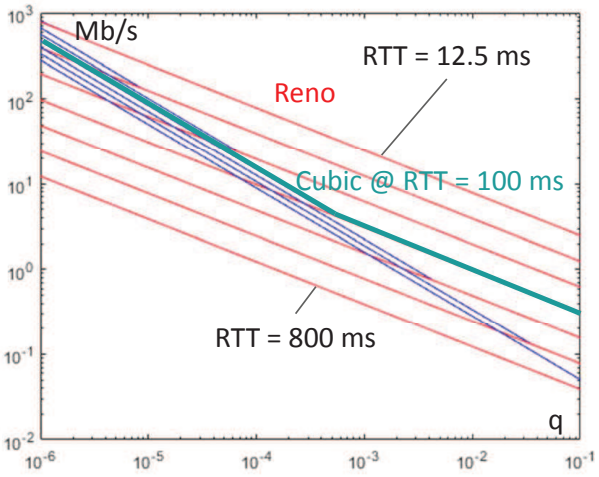


Figure 2.11: The throughput of CUBIC, as predicted by Equation (2.10), as a function of the loss probability, for exponentially increasing values of the round-trip time. The red parallel lines (in log-log scale) represent the throughput of TCP Reno. The throughput of CUBIC is given by the blue line down to the point where it crosses the red line.

This formula is a bit coarse in that it ignores mixed cases, where CUBIC spends part of its time in the TCP friendly region, part in the concave region. This is visible in the knee of every curve in Figure 2.11, which occurs at the point where the max in Equation (2.10) goes from one branch to the other. At such points the formula is not accurate. As with TCP Reno, the throughput of CUBIC decreases with the round trip time; however, for large round trip times, the dependency is less pronounced.

2.3.2 ACTIVE QUEUE MANAGEMENT

THE BUFFERBLOAT SYNDROM. This is a non desirable side-effect of using loss as congestion indication. Consider, for example, a scenario where a number of sources share a bottleneck link and use loss based congestion control (such as TCP Reno or CUBIC). Assume sources increase their window size more or less simultaneously. Figure 2.12 shows the evolution of the rate at which every source delivers packets to destination and of the round trip times. At the beginning, when windows are small, the rate is limited by the window and is equal to the window divided by round trip time; in this regime there is hardly any queuing and the round-trip time is constant equal to $RTT_{\min}$. As windows increase, the sum of source rates attains the link capacity (point A on Figure 2.12) and queuing is still small. At this point, since the buffer is very large, sources do not experience any loss and continue to increase their windows, up to point B. From point A to point B the queuing delay and hence the queuing delay increase, and the rate of delivery of packets to the destination remains the same, as it is determined by the bottleneck link. At point B the link buffer fills up; beyond point B, the link buffer starts to overflow, sources experience losses and the window (in average) does not increase any more.

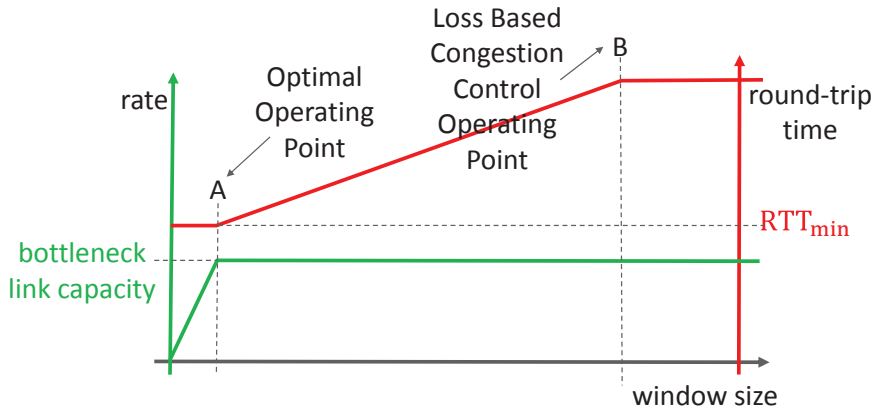


Figure 2.12: Operating point of congestion control when there is a single bottleneck link and the buffer is very large. Adapted from [5].

Point B is where loss-based congestion control operates in steady state for this scenario: there, the large link buffer is constantly oscillating from almost full to full and the round trip time is large. The buffer is not well utilized since it is constantly very full, which translates into large round trip times. In contrast, it would be better to operate around point A: there, the delivery rate is the same but the round trip time is much less. This discussion illustrates that, with loss-based congestion control, large buffers might be harmful. However, large buffers are needed for bursty traffic and do help avoid losses when there are temporary overloads due to bursts, not due to sustained congestion.

RANDOM EARLY DETECTION (RED). The root cause of buffer bloat is that the congestion indicator is packet drop, which occurs only when buffers are full ("tail drop"). A mitigation method, called "active queue management", is therefore to drop packets in a network buffer well before the buffer is full. It replaces tail drop by an intelligent admission decision for every incoming packet, based on a local algorithm. The algorithm uses a estimator of the long term load, or queue length;

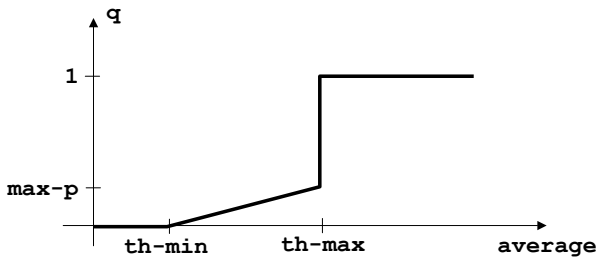


Figure 2.13: Target drop probability as a function of average queue length, as used by RED

in contrast, tail drop bases the dropping decision on the instantaneous buffer occupancy only. The most widespread algorithm is 'Random Early Detection" (RED) [3], which works as follows.

- For every incoming packet, an estimator `avg` of the average queue length is computed. It is updated at every packet arrival, using exponential smoothing:
  ```
  avg :=  a * measured queue length + (1 - a) * avg
  ```
  where a is the smoothing parameter, between 0 and 1.
- The incoming packet is randomly dropped, according to:
  ```
  if avg <  th-min accept the packet
  else if th-min < avg < th-max drop packet
         with probability p explained below
  else if th-max <= avg drop the packet
  ```
  where `th-min` and `th-max` are thresholds on the buffer content. By default, data is counted in packets.

The drop probability p is computed in two steps. First, the target drop probability q is computed by using the function of `avg`, shown on Figure 2.13, which, for `avg` between `th-min` and `th-max`, is equal to

```
q =  max-p * (avg - th-min) / (th-max - th-min)
```

Second, the *uniformization* procedure is applied, as described now. We could simply take $p := q$. This would create packet drop events such that the interval between packet drops would tend to be geometrically distributed (assuming q varies slowly). The designers of RED decided to have smoother than geometric drops. Specifically, they would like that this interval is uniformly distributed over $\{1,2,...,\frac{1}{q}\}$, assuming that $\frac{1}{q}$ is integer.

This can be achieved using the *hazard rate* of the uniform distribution, which we introduce now. Let T be the random variable equal to the packet drop interval, i.e. $T = 1$ if the first packet following the previous drop is also dropped. The hazard rate $p(k)$ of T is the probability that a packet is dropped, given that there were $k-1$ packets since the previous packet drop occurred, i.e. $p(k) = \mathbb{P}(T = k|T \geq k)$. If T has a geometric distribution, $p(k)$ is independent of k. In contrast, if T is uniformly distributed between 0 and a, then $p(k) = \frac{1}{a-k+1}$, so that $p(k)$ increases when

k increases from 1 to a and $p(a) = 1$. Furthermore, it can easily be shown that if we want drop packets such that the interval between packet drops has a specific distribution, it is sufficient to record the number k elapsed since the last packet drop and to drop a packet with probability $p(k)$. This is why RED computes `p` by letting

```
p = q/(1 - nb-packets * q)
```

where `nb-packets` is the number of packets accepted since the last drop.

When RED is used with proper tuning of its parameters, bufferbloat can be reduced: large buffers are used to absorb bursts but in average are not very full. Another effect is that long-lived flows experience a drop probability which depends only on the average amount of traffic present in the network, not on short term traffic fluctuations.

Last, a modified version of RED can be used to provide differentiated drop probabilities: a provider may modify the computed value of `p` in order to give larger drop probabilities to some flows, e.g. in order to give preference to its internal video streaming flows over external flows. This *non neutral* behaviour is against the original ideas of the Internet, but is technically possible and probably in-use today.

2.3.3 EXPLICIT CONGESTION NOTIFICATION

The Internet uses packet drops as the negative feedback signal for end-to-end congestion control. Losses have a negative impact on the delay performance of TCP as the lost packets need to be retransmitted. A more friendly mechanism is to use an explicit congestion signal, as was originally done with Jain's "Decbit". In the Internet, this is called Explicit Congestion Notification (ECN).

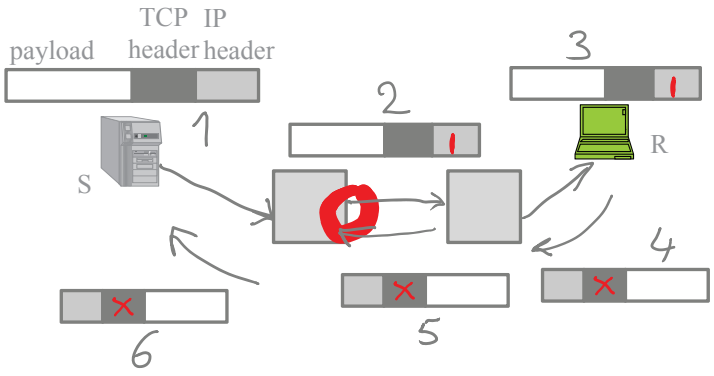


Figure 2.14: ECN uses a combination of fields in the IP and TCP headers

ECN works with TCP and uses a combination of fields in the IP and TCP headers. With ECN, when a router experiences congestion, it marks a "Congestion Experienced" bit in the IP header (red bars in Figure 2.14). A TCP destination that sees such a mark in received packet sets the ECN Echo (ECE) flag in the TCP header of packets sent in the reverse direction (red crosses in Figure 2.14). When a TCP source receives a packet with the ECE flag from the reverse direction, it performs multiplicative decrease (e.g. reduces the window by 0.5 for TCP Reno, by 0.7 for

CUBIC). The source then sets the Congestion Window Reduced (CWR) flag in the TCP headers. The receiver continues to set the ECE flag until it receives a packet with CWR set. The effect is that multiplicative decrease is applied only once per window of data (typically, multiple packets are received with ECE set inside one window of data).

ECN requires an active queue management such as RED to decide when to mark a packet with Congestion Experienced: instead of dropping a packet with the probability computed by RED, it marks it. If all parameters are properly set, a network with only ECN flows can avoid dropping any packet due to congestion in router buffers. This considerably increases the delay performance of every source.

An ECN-capable router needs to known whether a TCP source responds to ECN or not; if not, the router should drop a packet instead of marking it since otherwise the source will not react. This is achieved by using a bit in the IP header to indicate whether a source does support ECN or not; to avoid misconfigurations or frauds, it is combined with a "Nonce Sum" mechanism (see RFC 3540 for details).

2.3.4 DATA CENTER TCP

Data Center TCP (DCTCP) is a variant of TCP congestion control that is adapted to the TCP traffic seen in data centers. There, typically, there is a coexistence of many short flows with low latencies (user queries, consensus protocols) and *jumbo* flows, i.e., with huge volume (backups, data base synchronization). One issue is the throughput performance of jumbo flows. In order to avoid packet losses, ECN is used; a remaining issue is the oscillations of the window due to the multiplicative decrease present in TCP Reno or CUBIC, which occurs for every congestion indication received (Note that for the conditions in a data center, where round-trip time is of the order of 10 microseconds, CUBIC behaves as TCP Reno). As illustrated in Figure 2.8, if W is the window size at which congestion occurs, the actual window oscillates between $0.5W$ and W (for TCP Reno) and is thus equal to $0.75W$ in average. The goal of DCTCP is to make the average window size very close to W.

To this end, DCTCP modifies TCP Reno as follows:

- ECN must be used. A DCTCP source monitors the probability of congestion. In order to do this, the behaviour of ECN is modified such that a TCP receiver marks TCP acknowledgements with ECE flags in proportion to the number of received packets with Congestion Experienced marks. The receiver estimates the congestion probability as the proportion of acknowledgement packets with ECE flags. This differs from standard ECN behaviour, where feedback is a single bit of information per round trip time.
- When there is some non zero probability of congestion q, the multiplicative decrease factor is
$$\beta_{\text{DCTCP}} = \left(1 - \frac{q}{2}\right)$$

It follows that if there is little congestion (q is small) then the window reduction is small, much smaller than 0.5.

It follows that DCTCP competes unfairly with other TCPs; it cannot be deployed outside data centers (or other controlled environments). Inside data centers, care must be given to separate the DCTCP flows (i.e. the internal flows) from other flows. This can be done with class based queuing, as we discuss later.

2.4 TCP FRIENDLY APPLICATIONS

It can no longer be assumed that the bulk of long lived flows on the Internet is controlled by TCP only: some applications such as videoconferencing often use UDP as they have stringent delay requirements. Other applications use QUIC, an application layer framework that runs on top of UDP and replaces TCP (and the secure socket layer).

The solution which is advocated by the IETF is that *all TCP/IP applications which produce long lived flows should mimic the behaviour of a TCP source*. We say that such applications are "TCP friendly". In other words, all applications, except short transactions, should behave, from a traffic point of view, as a TCP source. For applications that use QUIC this is simple, as QUIC uses packet numbering and acknowledgements and reproduces the same congestion control features as TCP.

But for applications that do not use acknowledgements, how can TCP friendliness be defined ? One solution is the TCP-Friendly Rate Control protocol [8], which works as follows.

1. the application determines its sending rate using an adaptive algorithm;
2. the applications is able to provide feedback in the form of amount of lost packets; the loss ratio is the feedback provided to the adaptive algorithm;
3. in average, the sending rate should be the same as for a TCP Reno connection experiencing the same average loss ratio. This is typically achieved by using the loss-throughput formula in Equation (2.5).

There are many possible ways to implement an adaptive algorithm satisfying the above requirements. This can be achieved only very approximately; for more details on equation based rate control, see [26].

TCP friendly applications often use the so-called "Real Time transport Protocol" (RTP). RTP is not a transport protocol like TCP; rather, it defines a number of common data formats used by multimedia application. RTP comprises a set of control messages, forming the so called RTP Control Protocol (RTCP). It is in RTCP that feedback about packet loss ratio is given. Then it is up to the application to implement a TCP friendly rate adaptation algorithm.

2.5 CLASS BASED QUEUING

In general, all flows compete in the Internet using the congestion control method of TCP (or are TCP-friendly). In controlled environments (e.g. a data center, a smart grid, a TV distribution network, a cellular network) the competition can be modified by using per-class queuing.

With per class queuing, routers classify packets (using an access list) and every class is guaranteed a minimum rate – classes may exceed the guaranteed rate by borrowing from other classes if there is spare capacity. This is enforced in routers by implementing one dedicated queues for every class; the arbitration between classes is performed by a scheduler which implements some form of weighted fair queuing. With weighted fair queuing, every class, say i, at this router, is allocated a weight $w_i > 0$ and the scheduler serves packets on an outgoing link in proportion of the weights. As a result, class i is guaranteed to receive a long-term rate equal to $r_i = c\frac{q_i}{\sum_i q_j}$, where c is the bit rate of the outgoing link and the summation is over all classes that share this link. For a concrete example of scheduler, see Deficit Round Robin [25]. If class i is using less than its guaranteed rate r_i, the unused capacity is available to other classes, also in proportion to their weights.

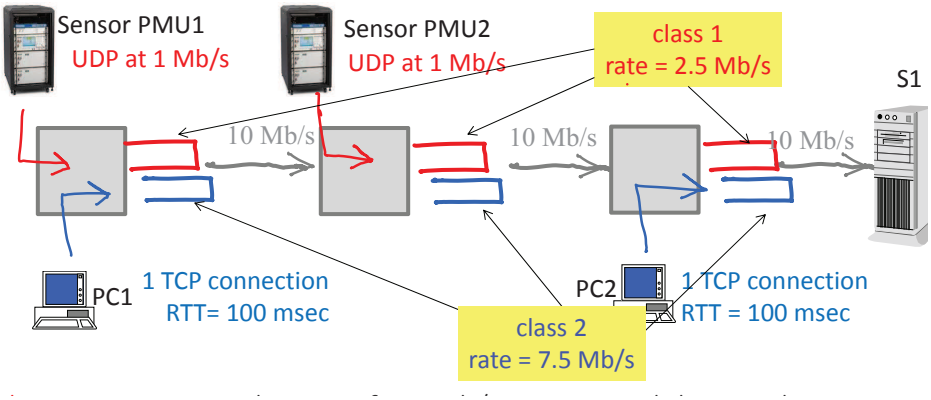


Figure 2.15: A Network with Per-Class Queuing.

Figure 2.15 illustrates a typical use of per-class queuing. Class 1 is for traffic sent by sensors at a constant rate; it is not congestion controlled and thus could cause congestion collapse. This is avoided here by allocating a sufficient rate to class 1 in every router, during a procedure called "traffic engineering". Class 2 is ordinary TCP/IP traffic, which is congestion controlled. It is also allocated a rate at every router. The rate at which sources of class 2 may send depends on the state of other sources in classes 1 and 2. Assume for example that the sources are exactly as shown on the figure. The two TCP connections share the available capacity, which is 9 Mb/s on the leftmost link and 8 Mb/s on the other links. If their RTTs are identical, each of these TCP connections will achieve a rate of 4 Mb/s. This illustrates how the rate that is allocated to class 1, but is not entirely used, is made available to class 2. Observe that class 2 is guaranteed a rate of 7.5 Mb/s. In normal operation, it obtains more; in contrast, if there is a failure in some of the class 1 devices that causes them to send more traffic than they normally should, class 2 is guaranteed to receive at least 7.5 Mb/s in total: the schedulers achieve isolation of the classes.

2.6 SUMMARY

1. Congestion control in the Internet is performed primarily by TCP, using the principles of Additive Increase, Multiplicative Decrease.
2. Congestion control in the internet provides a form of fairness close to proportional fairness, but with a non-desired bias against long round trip times.
3. The historical version of congestion control was introduced in TCP-Reno. Its loss-throughput formula maps the packet loss probability to the achieved rate.
4. Many variants of TCP congestion control exist. CUBIC is one of them, it replaces the linear window increase by a cubic function in long fat networks.
5. With Explicit Congestion Notification, routers signal losses to TCP sources without dropping packets.
6. RED is a mechanism in routers to drop packets before buffers get full. It avoids buffer to fill up unnecessarily when the traffic load is high. A variant of it can also be used to compute the congestion signal sent by routers when ECN is used.
7. Data Center TCP replaces the constant multiplicative decrease factor by a factor that depends

on an estimation of the loss or congestion probability. It is more aggressive than regular TCP (is not TCP-friendly) and is typically deployed in closed environments.

8. Non TCP applications that send large amounts of data should be TCP-friendly, i.e. not send more than TCP would. This can be achieved by mimicking TCP or by controlling the rate and enforce TCP Reno's loss throughput formula.
9. Per-class queuing is used in closed environments to support non TCP-friendly traffic.

Bibliography

[1] A. Benveniste, M. Metivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations*. Springer Verlag, Berlin, 1990.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Patridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. *IETF RFC 2309*, April 1998.

[4] L. Brakmo and L. Petersen. Tcp vegas. *IEEE Trans. on Networking*, October 1995.

[5] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50, 2016.

[6] Chiu D.M. and Jain R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[7] S. Floyd. Connections with multiple congested gateways in packet switched networks, part 1: one way traffic. *ACM Computer Communication Review*, 22(5):30–47, October 1991.

[8] S. Floyd, M Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. *Internet Request for Comments RFC5348*, 2008.

[9] Sally Floyd. Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic. *Computer Communication Review (also available from: ftp://ftp.ee.lbl.gov/papers/gates1.ps.Z)*, 21(5):30–47, October 1991.

[10] S. Golestani and S. Bhattacharyya. End-to-end congestion control for the internet: A global optimization framework. *Proc of ICNP, Oct 98*, 1998.

[11] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[12] E. L. Hahne. Round robin scheduling for max-min fairness in data networks. *IEEE JSAC*, pages 1024–1039, September 1991.

[13] Thomas R. Henderson, Emile Sahouria, Steven McCanne, and Randy H. Katz. On improving the fairness of tcp congestion avoidance. In *Proc. of the IEEE GLOBECOM '98*, Sydney, Australia, November 1998. IEEE.

[14] F.P. Kelly, A. K. Maulloo, and D.K.H. Tan. Rate control for commmunication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49, 1998.

[15] H. J. Kushner and D.S. Clark. Stochastic approximations for constrained and unconstrained systems. *Applied Mathematical Sciences*, 26, 1978.

[16] Liung L. Analysis of recursive stochastic algorithms. *IEEE transactions on automatic control*, 22:551–575, 1977.

[17] T. V. Lakshman and U. Madhow. The performance of tcp for networks with high bandwidth delay products and random loss. *IEEE/ACM Trans on Networking*, 5(3):336–350, June 1997.

[18] L. Massoulie and J. Roberts. Fairness and quality of service for elastic traffic. In *Proceedings of Infocom*, 1999.

[19] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behaviour of the tcp congestion avoidance algorithm. *Computer Communication Review*, 3, July 1997.

[20] R.K. Miller and A. N. Michell. *Ordinary Differential Equations*. Academic Press, New-York, 1982.

[21] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. In *SPIE'98 International Symposium on Voice, Video and Data Communications*, October 1998.

[22] Mazumdar R., Mason L.G., and Douligeris C. Fairness in network optimal flow control: Optimality of product form. *IEEE Transactions on Communication*, 39:775–782, 1991.

[23] B. Radunović and J.-Y. Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on Networking (TON)*, 15(5):1073–1083, 2007.

[24] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. CUBIC for Fast Long-Distance Networks. *Internet Request for Comments RFC8312*, 2018.

[25] Madhavapeddi Shreedhar and George Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking*, 4(3):375–385, 1996.

[26] M. Vojnovic and J.-Y. Le Boudec. On the long-run behavior of equation-based rate control. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, August 2002.

[27] Peter Whittle. *Optimization under Constraints*. Wiley and Sons, Books on Demand, `www.umi.com`, 1971.