Chapter 7

Worst-Case Event Systems

In many application domains events are not Poisson distributed. For some applications it even makes sense to (more or less) assume that events are distributed in the worst possible way (e.g. in networks, packets often arrive in bursts). In this section we study systems from a worst-case perspective. In particular, we analyze the price of not being able to foresee the future. This is a phenomenon that often occurs in discrete event systems (such as the Internet), but also in our daily life. This area of research is often referred to as Online Algorithms.

7.1 Ski Rental

We start out with a seasonal “toy example,” ski rental. Imagine that you want to start a new hobby (e.g. skiing, skateboarding, having a boy- or girlfriend), but you don’t yet know whether you will like it. The equipment is expensive, therefore you decide to first rent it a few times, before you buy (or get married?). When dealing with this problem, we (informally speaking) assume that Murphy’s law will strike: as soon as you buy, you will lose interest in the subject. Arguments like “I rented skis 17 times, and like it so much that I will go skiing for at least 1717 more times” do not count in Murphy’s world. Instead, once you buy skis you can be sure to meet new friends, and they think that skiing is for losers, and snowboarding or whatever is the new hip thing.

We first radicalize the problem (to make it mathematically more elegant and tractable):

Definition 7.1 (Ski Rental). The ski rental problem consists of two values:
- Input: a real number \( u \geq 0 \), representing the time a skier will end up skiing, chosen by an adversary.
- Algorithm: a real number \( x \), at which the algorithm will stop renting skis, and instead buys skis for price 1.

Remarks:
- The algorithm does not know the input \( u \).
- The algorithm is represented by a single value. This is rather unusual.

The cost of the algorithm with value \( x \) on input \( u \) is \( \text{cost}(x) \):

\[
\text{cost}(x) = \begin{cases} 
    u & \text{if } u \leq x \\ 
    x + 1 & \text{if } u > x 
\end{cases}
\]

The goal is to develop an algorithm \( x \) that is good for any input \( u \). We compare the cost of the algorithm with the cost of an optimalclairvoyant ("offline") algorithm:

\[
\text{cost}_{opt}(u) = \begin{cases} 
    u & \text{if } u \leq 1 \\ 
    1 & \text{if } u > 1 
\end{cases} = \min(u, 1).
\]

Definition 7.2 (Competitive Analysis). An online algorithm \( A \) is \( c \)-competitive if for all finite input sequences \( I \)

\[
\text{cost}_{A}(I) \leq c \cdot \text{cost}_{opt}(I) + k,
\]

where \( \text{cost} \) is the cost function of the algorithm \( A \) and the optimal offline algorithm, respectively, and \( k \) is a constant independent of the input. If \( k = 0 \), then the online algorithm is called strictly \( c \)-competitive.

Theorem 7.3. Ski rental is strictly 2-competitive. The best algorithm is \( x = 1 \).

Proof. When looking at strictly competitive ski rental algorithms, we can equivalently ask for

\[
\frac{\text{cost}(x)}{\text{cost}_{opt}(x)} \leq \epsilon,
\]

Let us investigate \( x = 1 \) in the ski rental algorithm. Then,

\[
\frac{\text{cost}(u)}{\text{cost}_{opt}(u)} = \begin{cases} 
    u \leq x = 1 \\ 
    u > x = 1 \\
\end{cases}
\]

Thus, the worst case is \( u > x = 1 \), and the competitive ratio is 2.

Is this optimal?
- Let’s try \( x > 1 \): In this case the adversary might/will choose \( u = x + \epsilon \). Then, the cost ratio is

\[
\frac{\text{cost}(x)}{\text{cost}_{opt}(x)} = \frac{x + 1}{\min(x, 1)} > 2.
\]
7.2 Randomized Ski Rental

Let's look at an algorithm A that chooses randomly between two values, \( z_1 \) and \( z_2 \) (with \( z_1 < z_2 \)), with probabilities \( p_1 \) and \( p_2 = 1 - p_1 \). Then,

\[
\text{cost}_A(u) = \begin{cases} 
  u & \text{if } u < z_1 \\
  p_1 \cdot (z_1 + 1) + p_2 \cdot u & \text{if } z_1 < u \leq z_2 \\
  p_1 \cdot (z_1 + 1) + p_2 \cdot (z_2 + 1) & \text{if } z_2 < u
\end{cases}
\]

The adversary, being very evil, will still choose the worst possible inputs. Convince yourself that only \( u_1 = z_2 + \epsilon \) and \( u_2 = z_2 + 2\epsilon \) are sensible. Since the adversary does not see the random coin flip of the algorithm, it as well has to choose its inputs randomly, with probabilities \( q_1 \) and \( q_2 \), respectively. The situation is equivalent to game theory if you're ambitious you might want to compute the Nash equilibrium for this game.

For the sake of simplicity, we will assign the algorithm the values

\( z_1 = 1/2, z_2 = 3, p_1 = 2/3, p_2 = 1/3 \).

We have \( \text{cost}_A = \frac{\text{cost}_A}{p_1 + p_2} \). For \( q_1 = z_1 + 1 \) and \( q_2 = z_2 + 1 \),

\[
\text{cost}_A = p_1 (z_1 + 1) + p_2 (q_1 u_1 + q_2 u_2 + 1)
\]

In short,

\[
\text{cost}_A = p_1 (z_1 + 1) + p_2 (q_1 u_1 + q_2 (z_2 + 2) + 1)
\]

Using the values from above,

\[
\text{cost}_A = \frac{3}{2} + \frac{1}{2} (q_2 u_2 + 1) = \frac{9}{4} (1 - q_2 / 2)
\]

Therefore,

\[
\text{cost}_A = q_1 + 1/2 + q_2 \cdot 1 - q_2 / 2.
\]

Hence,

\[
\text{cost}_A = q_1 + 1/2 + q_2 \cdot 1 - q_2 / 2.
\]

Then, the expected competitive ratio is

\[
E[\text{cost}] = \frac{1}{2} \int_0^1 \int_0^1 \int_0^1 \frac{1}{u} \text{d}u \text{d}v \text{d}w = \ldots = 1.75.
\]
7.2. Randomized Ski Rental

Was that a valid argument? Why yes, why no?

We assumed that the adversary chooses \( u \) with uniform distribution. This is not OK. In this specific example, an adversary can cause much more harm by choosing values close to 1. In addition, it was not correct to sum up the ratios of the costs, instead we should compute the ratio of the expected costs.

Instead, we should rather solve

\[
E[c] = \int_0^1 \left( \int_0^u (x + 1)p(x)\,dx \right)\,du + \int_0^1 \left( \int_u^1 p(x)\,dx \right)\,du
\]

where \( p(x) \) is the probability distribution of the algorithm, and \( d(u) \) is the probability distribution of the adversary, with \( \int_0^1 p(x) = \int_0^1 d(u) = 1 \). The adversary chooses its distribution \( d(u) \) such that it maximizes the expected competitive ratio \( E[c] \), and the algorithm chooses its distribution \( p(x) \) such that it minimizes \( E[c] \).

This is a very hard task. However, we can tackle it by making the problem independent of the adversarial distribution. How does this work???

The idea is as follows: if the adversary chooses a value \( u \) with \( u \leq 1 \) then it incurs an optimal cost \( \text{cost}_A(u) = u \). If we want our algorithm to be strictly competitive, all we have to do is to incur a cost less than \( c \cdot u \) when being offered input \( u \), for all \( u \). In other words, we have to choose the algorithm's probability function \( p(x) \) such that \( \text{cost}_A(u) \leq c \cdot u \).

Recall that the algorithm's cost is

\[
\text{cost}_A(u) = \begin{cases} 
 u & \text{if } u \leq x \\
 x + 1 & \text{if } u > x
\end{cases}
\]

(7.1)

Again, it seems natural to restrict the algorithm to values between 0 and 1. Also the adversary can restrict itself to values between 0 and 1, because, if a value higher than 1 is presented, the adversary, and the algorithm infer exactly the same cost as if the value 1 was presented. Therefore,

\[
\int_0^1 (x + 1)p(x)\,dx + \int_0^1 u p(x)\,dx \leq c \cdot u, \quad \text{with} \quad \int_0^1 p(x)\,dx = 1.
\]

Having a hunch that the best probability function will probably be an equality, we immediately try

\[
\int_0^1 (x + 1)p(x)\,dx + \int_0^1 u p(x)\,dx = c \cdot u, \quad \text{with} \quad \int_0^1 p(x)\,dx = 1.
\]

We first differentiate with respect to \( u \), getting

\[
(u + 1)p(u) + \int_0^1 p(x)\,dx + u \int_0^1 p(x)\,dx = c.
\]

We again differentiate with respect to \( u \), and get

\[
\frac{dp(u)}{du} \cdot u + p(u) = 0 \quad \Rightarrow \quad \frac{dp(u)}{du} = -p(u).
\]

That's one of the few differential equations everybody knows:

\[
p(u) = \alpha \cdot e^{-u}.
\]

In order to reveal \( \alpha \) we use \( \int_0^1 p(x)\,dx = 1 \):

\[
1 = \int_0^1 \alpha e^{-u} \,du = \alpha (e^0 - e^{-1}) = \alpha = \frac{1}{e - 1}.
\]

In other words, \( p(u) = \frac{e}{e - 1} \). We insert \( p(u) \) into the first differentiation:

\[
\alpha = p(u) + \int_0^1 p(x)\,dx = \frac{e}{e - 1} + \frac{1}{e - 1} = \frac{e + 1}{e - 1}.
\]

Note that also for inputs \( u > 1 \) the inequality \( \text{cost}_A(u) \leq c \cdot \text{cost}_A(u) \) holds.

**Theorem 7.4.** In other words, with \( p(u) = \frac{e}{e - 1} \) we have an algorithm that is \( \frac{e}{e - 1} \)-competitive in expectation.

**Remarks:**

- The big question remains: Can we get any better???

7.3. Lower Bounds

Time to think about lower bounds. Lower bounds for randomized algorithms often use the Von Neumann / Yao Principle, which we state and use without proof:

**Theorem 7.5** (Von Neumann / Yao Principle). Choose a distribution over problem instances (for ski rental, e.g. \( d(u) \)). If for this distribution all deterministic algorithms cost at least \( c \), then \( c \) is a lower bound for the best possible randomized algorithm.

For ski rental we are in the lucky situation that we can easily parameters all possible deterministic algorithms by \( z \geq 0 \). Now we have to choose a distribution of inputs, with \( d(u) \geq 0 \) and \( \int d(u) = 1 \).

For example, \( d(u) = \frac{1}{2} \) for \( 0 \leq u \leq 1 \) and \( d(u) = 0 \) for \( u \geq 1 \).

Example algorithms:

- \( z = 0 \) (immediate buy): incurs a constant cost 1 for all possible input distributions. Therefore \( \text{cost}_{	ext{opt}}(d(u)) = 1 \).
- \( z = 1 \) (worst-case deterministic algorithm): incurs the same cost as the optimal offline algorithm for small \( u \) but cost 2 for \( u = \infty \) which happens with probability 1/2; when summing up we see that \( \text{cost}_{	ext{opt}}(d(u)) = 5/4 \).

More formally, the cost of the optimal offline algorithm is

\[
\text{cost}_{	ext{opt}}(d(u)) = \int_0^1 \left( \frac{e}{e - 1} + \frac{1}{2} \right) \,du = \frac{5}{4}.
\]
7.4. THE TCP ACKNOWLEDGEMENT PROBLEM

For general \( x \leq 1 \) the cost of the algorithm is

\[
\text{cost}_x = \frac{1}{2} \left( \int_0^1 u \, du + \int_1^x \left( \frac{1}{2}u + \frac{1}{2}(e^u - 1) \right) \, du \right) = \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right) - \frac{1}{2} \left( 1 + \frac{x}{2} + \frac{x^2}{2} \right) \geq 1.
\]

For general \( x > 1 \) the cost of the algorithm is

\[
\text{cost}_x = \frac{1}{2} \int_0^1 u \, du + \frac{1}{2} \left( e^1 - 1 \right) - \frac{1}{4} \left( x + 1 \right) > \frac{1}{4} + \frac{x + 1}{2} > \frac{5}{4}.
\]

Using \( \text{cost}_x(\delta(u)) = 8/4 \) we conclude that the competitive ratio \( c \) is at least \( 4/3 = 1.33 \).

Remarks:

- Note that for distribution \( d(u) \) indeed \( x \to 0 \) is the best algorithm.
- The lower bound of 1.33 and the upper bound of 1.58 do not match.
- As argued above, the immediate buy algorithm is worst with very small \( u \). In order to make our lower bound stronger it could therefore be beneficial to tune the input distribution such that it contains more small \( u \) values.
- Guessing the right input distribution is indeed hard. However, similarly to the upper bound, this can be derived using differential equations. The worst input distribution is \( d(u) = \frac{1}{u^2} \), for \( 0 < u < 1 \), and \( d((u)) = 1/e \).
- Next, let us study some online problems in the Internet ("Web") context. We will discover surprising connections to ski rental.

7.4. The TCP Acknowledgement Problem

TCP is a layer 4 networking protocol of the Internet. It features, among other things:

- An error handling mechanism which tracks transmission errors and reordering of packets, using sequence numbers and acknowledgements.
- A "friendly" exponential slow start mechanism such that new connections do not overload the network.
- Flow Control: A sliding window sender/receiver buffer that simplifies handling and prevents the receiver buffer from overloading.
- Congestion Control: A back-off mechanism that should prevent network overloading.

In this first part we study the TCP Acknowledgement Problem. We study a single sender/receiver pair, where the sender sends packets and the receiver acknowledges them (without sending packets itself). There are several TCP implementations available, with various acknowledgement procedures. In order to save resources, no implementation sends acknowledgements right away. Instead, these implementations send cumulative acknowledgements ("I received all packets up to packet \( n \)). This mechanism is the subject of this section.

At the receive side, the situation looks like in Figure 7.2.

![Figure 7.2: TCP ACK problem](image)

**Definition 7.6 (TCP Acknowledgement Problem).** The receiver's goal is a scheme which minimizes the number of acknowledgements plus the sum of the latencies for each packet, where the latency of a packet is the time difference from arrival to acknowledgement. More formally, we have:

- \( n \) packet arrivals, at times: \( a_1, a_2, \ldots, a_n \)
- \( k \) acknowledgements, at times: \( t_1, t_2, \ldots, t_k \)
- And we want to minimize:

\[\min k \text{ s.t. } \sum_{i=1}^k \text{latency}(i), \text{ with latency}(i) = t_j - a_i, \text{ where } j \text{ such that } t_{j-1} < a_i \leq t_j.\]

Remarks:

- Note that in Figure 7.2 the total latency is exactly the area between the two curves.
- Clearly, we are comparing apples with oranges when comparing the number of acknowledgements with the sum of latencies. However, when scaling the time accordingly, this should not be a big problem.

One version of Sack, for example, always sends three before acknowledging in order to support multiple acknowledgements in a single message. In one version of RED, TCP-sack has a 50/50 handshake, and acknowledges all packets received so far.
7.4. THE TCP ACKNOWLEDGEMENT PROBLEM

- There are quite a few technical exceptions. In many implementations, signaling packets are usually acknowledged faster (e.g., SYN, FIN); also TCP standard wants implementations to acknowledge packets within 50ms. Since the receiver is usually also sender, it might also delay its own sending packets.
- In our studies we do not learn the future from the past. A machine learning approach could give a totally different perspective.

The $z = 1$ algorithm is sketched in Figure 7.3. Whenever a rectangle with area $z = 1$ does not intersect the two curves, the receiver sends an acknowledgment, acknowledging all previous packets.

![Figure 7.3: The $z = 1$ algorithm](image)

**Lemma 7.7.** The optimal algorithm sends an ACK between any pair of consecutive ACKs by algorithm with $z = 1$.

**Proof.** For the sake of contradiction, assume that, among all algorithms which achieve the minimum possible cost, there is no algorithm which sends an ACK between two ACKs of the $z = 1$ algorithm. We propose to send an additional ACK at the beginning (left side) of each $x = 1$ rectangle. Since this ACK saves latency 1, it compensates the cost of the extra ACK. That is, there is an optimal algorithm who chooses this extra ACK. \[ \Box \]

**Theorem 7.8.** The $z = 1$ algorithm is 2-competitive.

**Proof.** We have $\text{cost}_{\text{opt}} = k_{\text{opt}} + \text{latency}_{\text{opt}}$ and $\text{cost}_{z=1} = k_{z=1} + \text{latency}_{z=1}$.

Since the optimal algorithm sends at least one ACK between any two consecutive ACKs of $A_{z=1}$ (previous Lemma), we know $k_{z=1} \leq k_{\text{opt}}$.

Also, by definition (see Figure 7.4),

\[
\text{latency}_{z=1} = \text{latency}_{\text{opt}} + \text{latency}(z = 1 \text{ without opt}) - \text{latency}_{\text{opt}} \text{ without } z = 1
\leq \text{latency}_{\text{opt}} + \text{latency}(z = 1 \text{ without opt})
\]

Using $\text{latency}(z = 1 \text{ without opt}) < k_{\text{opt}} - 1$ (if any of these rectangles were of size 1 or larger, $A_{z=1}$ would have ACKed earlier) we get:

\[
\text{cost}_{z=1} = k_{z=1} + \text{latency}_{z=1}
\leq k_{\text{opt}} + \text{latency}_{\text{opt}} + \text{latency}(z = 1 \text{ without opt})
\leq k_{\text{opt}} + \text{latency}_{\text{opt}} + k_{\text{opt}} - 1
= 2(k_{\text{opt}} + \text{latency}_{\text{opt}}) \leq 2 \cdot \text{cost}_{\text{opt}}
\]

**Remarks:**
- It's no coincidence that we called the algorithm $z = 1$. Similarly to ski rental, it is possible to choose any $z$. In fact, if you really think about it, the TCP ACK problem is in fact very much like ski rental! Indeed, if you wait for a rectangle of size $x$ with probability $p(x) = \frac{1}{x}$, you end up with a randomized TCP ACK solution which is $\frac{1}{\ln(2)}$ competitive in expectation.
- Many other problems are also just like ski rental! That's why we studied it in the first place: E.g., the Halftaxi problem (originally known as the Halftaxi problem). Buying a HalfTaxi with which reduces each trip by $\beta$ is $\frac{1}{\beta}$ competitive.

7.5 The TCP Congestion Control Problem

As an example, we study the sender side of TCP. We ask: How many segments can a sender inject into the network without overloading it? The problem is that a sender does not know the current bandwidth between itself and the receiver. And, more importantly, this
7.6 THE STATIC MODEL

bandwidth might change over time with other connections starting up, or closing down.

Here's our model:

- We divide the time into periods (or slots).
- In each period, there is an unknown threshold \( u \), where \( u \) is the number of packets (or segments, or bytes) that could successfully be transmitted from sender to receiver, without overloading the network.
- In period \( t \), the sender chooses to transmit \( x_t \) packets.
- If \( x_t \leq u \), we are fine. However, sending at too conservative or small rates \( x_t < u \) is a waste of the available bandwidth. One possible way to capture this aspect would be to use an opportunity cost function of the form \( c(x_t) = u - x_t \).
- If \( x_t > u \), we are not fine. We are overloading the channel. There are several cost models possible. In a severe cost model, nothing gets transmitted (cost, \( c(x_t) = u \)), in a less severe cost model, some fraction of the packets might get dropped (e.g., \( c(x_t) = a(x_t - u) \)).

7.6 The Static Model

We start out with the simplest possible model, where the bandwidth is constant over time, that is, \( u_t = u \). The problem is then to find the correct bandwidth \( u \) (with something like binary search); once the sender finds the correct bandwidth, there will be no more cost. We assume first that \( u \) is an integer, and that \( 1 \leq u \leq n \), that is, there is an upper bound \( n \) for the bandwidth.

Possible algorithms:

- Plain old binary search needs \( \log_2 n \) search steps. For a worst-case choice of \( u \) the algorithm will often inject too many packets, and in a severe cost model, have cost \( c(x_t) = O(n) \) in most steps, thus the total cost is \( O(n \log n) \).

A standard TCP congestion control mechanism is usually following the AIMD (Additive Increase Multiplicative Decrease) paradigm: Once TCP sends as many packets as the network can handle, it starts dropping packets. The sender can witness this (with missing ACKs), and consequently decreases its transmission rate (for example in a multiplicative way, e.g., by a factor 2). Then the sender starts increasing its transmission rate again, but slowly, to approach the "right" bandwidth again (for example by 1, in an additive way). In our model, if the real bandwidth is \( u < n - 1 \), such an algorithm will clearly be very much off the right bandwidth \( u \) most of the time. Since approaching \( u \) takes \( O(n) \) steps, and in the severe cost model most steps cost \( c(x_t) = O(n) \), the cost of the AIMD algorithm is \( O(n^2) \).

The obvious question. Can we do better???

56 CHAPTER 7. WORST-CASE EVENT SYSTEMS

The algorithm operates on a partition interval \([i, j]\), originally \([i, j] = [1, n] \). The algorithm has two phases:

- Phase 1: Find the right power-two-up bound \( j \), that is, find \( j \) such that \( 2^s < j \leq 2^{s+1} \) by testing \( 2^s + 1 \leq j \). If \( 2^s + 1 \leq \max(i, j) \) phase 2 begins, else set \([i, j] = [1, 2^s]\) and stay in phase 1.

- Phase 2: We are given \([i, j]\) with \( 2^{s-1} + 1 \leq i < j \leq 2^s \). Now we test \( i + \max\left(1, \frac{2^{s-1}}{2^{s-2} + 1}\right) \) with \( m \) being the largest integer such that \( j - i \leq \frac{1}{2^m} \). Then adapt \([i, j]\) accordingly.

Remarks:

- It can be shown that the cost of the Shrink algorithm is \( O(n \log \log n) \).

- For large \( n \), it is remarkable that the vast majority of increase steps are increments by just 1. And almost all decrease steps are substantial. In other words, the algorithm is an AIMD algorithm.

- If \( n \) is not known, we can find an upper bound of \( u \) quickly by a repeated squaring technique first, that is, test 2, then 2^2 = 4, then 2^4 = 16, then 2^8 = 256, etc. It can be shown that the total cost is \( O(n \log \log n) \).

- There is a lower bound of \( O(n \log \log n \log \log \log n) \). Hence the Shrink algorithm is asymptotically almost optimal.

- However, this was only an upper bound example. What we are really interested in are dynamic models.

7.7 The Dynamic Model

In this section, the threshold may vary from step to step, i.e., the adversary chooses a sequence \( u_t \). Then, the adversary knows the algorithm's sequence \( x_t \). The problem is in advance. Clearly, we are again in the realm of online algorithms and competitive analysis.

We have postulated that \( c_{opt}(t) \leq c_{opt}(t) \). Observe that an optimal offline algorithm knowing the input (as in ski rental or TCP ACK) can always play \( x_t = u_t \), which implies that \( c_{opt} = 0 \). No online algorithm can be competitive.

For this reason it seems more fruitful to look at gain (or profit) rather than cost. We update our definition from ski rental as follows:
7.8 Bandwidth in a Fixed Interval

Definition 7.9 (Competitive Analysis). An online algorithm $A$ is strictly $c$-competitive if for all finite input sequences $I$

$$
cost_A(I) \leq c \cdot \text{cost}_{opt}(I), \quad \text{or}$$

$$
cost_A(I) \geq \text{gain}_{opt}(I).$$

Remarks:

- Note that in both cases $c \geq 1$. The closer $c$ is to 1, the better is an
algorithm.

For a severe cost model, a natural definition of gain could look as follows:

$$\text{gain}_{opt}(u) = \begin{cases} x_i & \text{if } x_i \leq u \\ 0 & \text{if } x_i > u \end{cases}$$

However, note that our adversary is too strong because knowing the algorithm it can always present an $u_i = x_i$ (or, if $x_i = 0$, any $u_0$). The total gain of the algorithm (given as $\sum_1 \text{gain}_{opt}(u)$) is 0. Therefore we need to further restrict the power of the adversary. Several restrictions seem to be reasonable and interesting:

- Bandwidth in a fixed interval: $u_i \in [a, b]$
- Multiplicatively (or additively) changing bandwidth: $u_i/\mu \leq u_{i+1} \leq \mu \cdot u_i$
- (or $u_i - a \leq u_{i+1} \leq u_i + a$)
- Changes with bursts

In the following, the three restrictions will be studied in turn.

7.8 Bandwidth in a Fixed Interval

We start out by letting the adversary choose $u_i \in [a, b]$. The algorithm is aware of the upper bound $b$ and the lower bound $a$. We first restrict ourselves to deterministic algorithms. In this case, note the following:

- If the deterministic algorithm plays $x_i > a$ in round $i$, then the adversary plays $u_i = a$.
- Therefore the algorithm must play $x_i = a$ in each round in order to have at least gain $a$.
- The adversary knows this, and will therefore play $u_i = b$
- Therefore, $\text{gain}_{opt} = a$, $\text{gain}_{opt} = b$, competitive ratio $c = b/a$.

As usually, we ask whether randomness might help? Let's try the ski rental trick immediately! In particular, for all possible inputs $u \in [a, b]$ we want the same competitive ratio:

$$c \cdot \text{gain}_{opt}(u) = \text{gain}_{opt}(u) = u.$$
Remarks:

- Great, upper and lower bound are tight!
- Didn't we ask for, w, z being integers? In this case, \( c = 1 + H_n - H_{n_1} \), where \( H_n \) is the harmonic number \( n \) defined as \( H_n = \sum_{i=1}^{n} \frac{1}{i} \).
- Note that the above cases where the bandwidth smoothly changes over time and does not jump up and down like crazy.

7.9 Multiplicatively Changing Bandwidth

Now the adversary must choose \( u \) such that \( u/\mu < u_{n_1} < \mu u \). The algorithm knows the maximal possible change factor \( \mu \) per round. We assume that the algorithm always uses the initial threshold \( u_0 \). Think of \( \mu \) as being a value such that the bandwidth changes a few percent only per period.

If the adversary keeps raising \( u \) as fast as possible (\( u_{n+1} = \mu \cdot u \) for several rounds), then it seems reasonable that the algorithm does the same. In particular, if the algorithm chooses \( u_{t+1} = (1 + \epsilon)u_t \), then

\[
\lim_{t \to \infty} \frac{u_t}{u_0} = \left( \frac{1 + \epsilon}{\mu^t} \right) = \infty.
\]

Therefore, if there was a successful transmission in period \( t \), the algorithm chooses \( u_{t+1} = \mu u_t \). On the other hand, if \( u_t \) was not successful, \( u_{t+1} = \lambda u_t \). We will set \( \lambda = 1/\mu^t \). The idea is that at least every other round is successful.

**Lemma 7.12.** After a non-successful round there is always a successful round.

**Proof.** Since we know \( u_1 \), the algorithm can choose \( u_1 = u_0 \), and have a success. Our invariant is that every non-successful round is followed by a successful round. Assume, for the sake of contradiction, that round \( t + 1 \) is the first non-successful round which follows after a non-successful round \( t \), which (by induction hypothesis) follows a successful round \( t - 1 \) (note that \( x_{t-1} \leq u_{t-1} \)). Since \( u_{t-1} \leq u_t/\mu \) (for all \( t \)), we have \( u_{t+1} \leq u_{t+1}/\mu \). On the other hand, we have \( u_{t+1} = \lambda u_t = \lambda u_{t-1} = x_{t-1}/\mu \). Therefore, \( x_{t+1} = x_{t-1}/\mu \leq u_{t+1}/\mu \leq u_{t+1} \), hence round \( t + 1 \) is a success. We have a contradiction, which proves that there can be only one non-successful round in a row.

**Lemma 7.13.** A successful round in \( \mu \)-competitive.

**Proof.** If a successful round \( t + 1 \) follows a successful round \( t \), round \( t + 1 \) is at least as competitive as round \( t \) since the algorithm set \( x_{t+1} = \mu x_t \).

- If a successful round \( t + 1 \) follows a non-successful round \( u < x_t \), then, since \( x_{t+1} = \lambda x_t \) and \( u_{t+1} \leq \mu u \), we have \( x_{t+1} = \lambda x_t \geq \mu u_{t+1}/\mu = u_{t+1}/\mu \).

**Theorem 7.14.** The algorithm is \((\mu = \mu)\)-competitive.

**Proof.** In a non-successful ("fail") round \( t \), it holds that \( u_t < \mu u_{t-1} \), because \( x_{t+1} \leq u_{t+1} \), (by Lemma 7.12), \( x_t = \mu x_{t-1} \) and \( u_{t+1} < \mu u_{t} \). Then

\[
\frac{\text{path}(\text{succ}) + \text{path}(\text{fall})}{\text{path}(\text{succ})} < \frac{u_t^\mu \cdot \text{path}(\text{succ}) + \mu \cdot \text{path}(\text{fall})}{\text{path}(\text{succ})} = \mu^t u.
\]

While this algorithm is good for small \( \mu \), the competitive ratio grows quickly for larger \( \mu \). In the following, we show that an algorithm which increases the bandwidth by a factor \( \mu \) after successful rounds and halves the rate after non-successful rounds is \( \mu \)-competitive.

**Theorem 7.15.** This new algorithm is \( \mu \)-competitive.

**Proof.** First, we show by induction that in each successful or good round, \( u_t \leq 2\mu u \). Put \( t = 1, u_1 = x_1 \), and the claim holds. For the induction step, consider the round \( t - 1 \) before the good round \( t \). There are two possibilities: either round \( t - 1 \) was unsuccessful or bad (\( x_{t-1} > u_{t-1} \)), or good (\( x_{t-1} \leq u_{t-1} \)). If round \( t - 1 \) was bad, we have \( x_{t-1} = u_{t-1}/\mu \) and \( u_{t-1} \leq u_{t-1} \leq u_{t-1}/\mu < \mu u_t \), hence \( u_t \leq 2\mu u_t \), and the claim holds. If on the other hand round \( t - 1 \) was good, the algorithm increases the bandwidth at least as much as the adversary.

Together with the induction hypothesis, the claim follows also in this case.

Having studied the gain in good rounds, we now consider bad rounds. We show that in the bad rounds following a good round \( t \), the adversary may increase the gain at most by \( 2\mu u_t \). Let \( x_{t+1} \geq u_{t+1} \geq u_{t+1}/\mu \). We know that \( x_{t+1} = \mu u_{t+1}/\mu \). By a geometric series argument, the gain of the adversary in the bad rounds is upper bounded by \( 2\mu u_t \).

Therefore,

\[
\mu = \frac{\text{path}(\text{succ}) + \text{path}(\text{fall})}{\text{path}(\text{succ})} < \frac{2\mu \cdot \text{path}(\text{succ}) + 2\mu \cdot \text{path}(\text{fall})}{\text{path}(\text{succ})} = 4\mu.
\]

7.10 Changes with Bursts

In the previous section, we assumed that the bandwidth changes by at most a given constant percentage \( \mu \) over time. However, one can imagine that in the real Internet there may be quiet times where the congestion level hardly changes, and times where there are very abrupt or bursty changes. In main objectives of this section is to present—without any analyses—adversary model which incorporates such a notion of bursts. Our model is based on concepts of network...
7.20. CHANGES WITH BURSTS

In order to analyze such bursty adversaries, similar techniques as those presented in Section 7.9 can be applied; we do not perform these computations here.