Theory of Renting Skis

- Scenario
  - you start a new hobby, e.g. skiing
  - you don’t know whether you will like it
  - expensive equipment: ~1 kFr

- 3 Alternatives
  - just buy a new equipment (optimistic)
  - always renting (pessimistic)
  - first rent it a few times before you buy (down-to-earth)

- You choose the pragmatic way, but Murphy’s law will strike!
  - first you rent, but as soon as you buy, you will lose interest in skiing

Overview: Worst-Case Analysis of DES

- Ski Rental
  - Randomized Ski Rental
  - Lower Bounds

- The TCP Acknowledgement Problem
- The TCP Congestion Control Problem
  - Bandwidth in a Fixed Interval
  - Multiplicatively Changing Bandwidth
  - Changes with Bursts

- Many application domains are not Poisson distributed!
  - sometimes it makes sense to assume that events are distributed in the worst possible way (e.g. in networks, packets often arrive in bursts)

Ski Rental Problem

- Expenses
  - buying: 1 kFr
  - renting: 1 kFr per month

- Scenario
  - first rent it for \( z \) months, then buy it
  - after \( u \) months you will lose your interest in skiing
  - 2 cases:
    \[ u \leq z \Rightarrow c^*(u) = u \text{ kFr} \]
    \[ u > z \Rightarrow c^*(u) = (z + 1) \text{ kFr} \]

- If you are a clairvoyant, then ...
  - \( u \leq 1 \) month \( \Rightarrow \) just renting is better \( \Rightarrow c^*(u) = u \text{ kFr} \)
  - \( u > 1 \) month \( \Rightarrow \) just buying is better \( \Rightarrow c^*(u) = 1 \text{ kFr} \)
  - \( \Rightarrow c^*(u) = \min(u, 1) \)
Competitive Analysis

• Definition
  An online algorithm \( A \) is \( c \)-competitive if for all finite input sequences \( I \)
  \[
  \text{cost}_A(I) \leq c \cdot \text{cost}_{\text{opt}}(I) + k
  \]
  where \( k \) is a constant independent of the input.
  If \( k = 0 \), then the online algorithm is called strictly \( c \)-competitive.

• When strictly \( c \)-competitive, it holds
  \[
  \frac{\text{cost}_A(u)}{\text{cost}_{\text{opt}}(u)} \leq c
  \]

• Example
  – Ski rental is strictly 2-competitive. The best algorithm is \( z = 1 \).

Randomized Ski Rental

• Deterministic Algorithm
  – has a big handicap, because the adversary knows \( z \) and can always present a \( u \)
    which is worst-case for the algorithm
  – only hope: algorithm makes random decisions

• Randomized Algorithm
  – chooses randomly between 2 values \( z_1 \) and \( z_2 \) (with \( z_1 < z_2 \))
    with probabilities \( p_1 \) and \( p_2 = 1 - p_1 \)
  – if \( u \leq z_1 \)
    \[
    \text{cost}_A(u) = p_1 \cdot (z_1 + 1) + p_2 \cdot u
    \]
  – if \( z_1 < u \leq z_2 \)
    \[
    \text{cost}_A(u) = p_1 \cdot (z_1 + 1) + p_2 \cdot (z_2 + 1)
    \]
  – if \( z_2 < u \)
    \[
    \text{cost}_A(u) = p_1 \cdot (z_1 + 1) + p_2 \cdot (z_2 + 1)
    \]
  – adversary chooses randomly
    – \( u_1 = z_1 + \varepsilon \) with probability \( q_1 \)
    – \( u_2 = z_2 + \varepsilon \) with probability \( q_2 = 1 - q_1 \)

• Example
  – \( z_1 = \frac{3}{5} \), \( z_2 = 1 \), \( p_1 = 2/5 \), \( p_2 = 3/5 \)
  – \( E[c] = \frac{\text{cost}_A}{\text{cost}_{\text{opt}}} = 1.8 \)

Randomized Ski Rental with infinitely many Values (1)

• Let \( r(u, z) \) be the competitive ratio
  for all pairs of \( u \) and \( z \)
• We are looking for the expected competitive ratio \( E[c] \)

• Adversary chooses \( u \) with uniform distribution
  \[
  E[c] = \frac{\int \int r(u, z) du dz}{\int du}
  \]
  \[
  = \frac{1}{2} + \int_{z=0}^{1} \int_{u=0}^{z} \frac{z+1}{u} du dz
  \]
  \[
  = 1.75
  \]

Randomized Ski Rental with infinitely many Values (2)

• Algorithm chooses \( z \) with probability distribution \( p(z) \)
  – it chooses \( p(z) \) such that it minimizes \( E[c] \)
• Adversary chooses \( u \) with probability distribution \( d(u) \)
  – it chooses \( d(u) \) such that it maximized \( E[c] \)
  \[
  E[c] = \frac{\int_0^1 \int_0^{z+1} p(z) d(u) du dz + \int_0^1 \int_0^u p(z) d(u) du dz}{\int_0^1 \int_0^{z+1} p(z) d(u) du dz + \int_0^1 \int_0^u p(z) d(u) du dz}
  \]
  \[
  \int p(z) = \int d(u) = 1
  \]
• How to find these probability distributions?
  – This is a very hard task!
  → We should make the problem independent of the adversarial distribution \( d(u) \).
Randomized Ski Rental with infinitely many Values (3)

- **Idea**
  
  Choose the algorithm’s probability function \( p(z) \) such that \( \text{cost}_z(u) \leq c \cdot \text{cost}_{\text{opt}}(u) \) for all \( u \)
  
  \( \rightarrow \) adversarial distribution \( d(u) \) doesn’t matter anymore

- \( \text{cost}_{\text{opt}}(u) = u \) for all \( u \) between 0 and 1

\[
\int_0^u (z + 1)p(z)dz + \int_u^1 u \cdot p(z)dz \leq c \cdot u
\]

with \( \int_0^1 p(z)dz = 1 \)

- Having a hunch: the best probability function \( p(z) \) will be an equality

  \( \rightarrow \) With \( p(z) = \frac{z^2}{c-1} \) we have an algorithm that is \( \frac{c}{c-1} \)-competitive in expectation.

Can we get any better???

- **Lower Bounds**

  - 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.

  - Ski Rental

    \( \rightarrow \) we are in a lucky situation, because we can parameterize all possible deterministic algorithms by \( z \geq 0 \)

    \( \rightarrow \) choose a distribution of inputs with \( d(u) \geq 0 \) and \( \int d(u) = 1 \)

  - Examples: \( d(u) = \frac{1}{2} \) for \( 0 \leq u \leq 1 \) and \( d(\infty) = \frac{1}{2} \)

    \( \rightarrow \) \( \text{cost}_z(d(u)) = 1 \)
    
    \( \rightarrow \) \( \text{cost}_{\infty}(d(u)) = 1 + z/2 - z^2/4 \geq 1 \)
    
    \( \rightarrow \) \( \text{cost}_z(d(u)) = 5/4 \)
    
    \( \rightarrow \) \( \text{cost}_{\infty}(d(u)) = 1/4 + (z + 1)/2 > 5/4 \)

    \( \rightarrow \) \( \frac{c}{\text{cost}_{\text{opt}}} = 1/\frac{1/4}{4} = 4/3 = 1.33 \)

TCP: Transmission Control Protocol

- Layer 4 Networking Protocol

  - transmission error handling
  
  - correct ordering of packets

  - exponential (“friendly”) slow start mechanism: should prevent network overloading by new connections

  - flow control: prevents buffer overloading

  - congestion control: should prevent network overloading

Packet Acknowledgment

- Sender

  - Sequence number in packet header

  - “Window” of up to \( N \) consecutive unack’ed packets allowed

  - ACK\( (n) \): ACKs all packets up to and including sequence number \( n \)

    \( \rightarrow \) a.k.a. cumulative ACK

    \( \rightarrow \) sender may get duplicate ACKs

    \( \rightarrow \) timer for each in-flight packet

    \( \rightarrow \) \text{timeout}(n): retransmit packet \( n \) and all higher seq# packets in window

  - already ack’ed

  - sent, not yet ack’ed

  - usable, not yet sent

  - not usable
The TCP Acknowledgment Problem

• Definition
  The receiver’s goal is to devise a scheme which minimizes the number of
  acknowledgments plus the sum of the latencies for each packet, where the
  latency of a packet is the time difference from arrival to acknowledgment.

• Given
  \( n \) packet arrivals, at times: \( a_1, a_2, \ldots, a_n \)
  \( k \) acknowledgments, at times \( t_1, t_2, \ldots, t_k \)
  latency(\( i \)) = \( t_j - a_i \) where \( j \) such that \( t_{j-1} < a_i \leq t_j \)

• Minimize
  \[ k + \sum_{i=1}^{n} \text{latency}(i) \]

The TCP Acknowledgment Problem: z=1 Algorithm (1)

• z = 1 Algorithm is: Whenever a rectangle with area \( z = 1 \) does fit
  between the two curves, the receiver sends an acknowledgement, acknowledging
  all previous packets.

The TCP Acknowledgment Problem: z=1 Algorithm (2)

• Lemma
  – 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 who 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
    \( z = 1 \) 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.

The TCP Acknowledgment Problem: z=1 Algorithm (3)

• Theorem: The \( z = 1 \) algorithm is 2-competitive.

• Similarity to Ski Rental
  – it’s possible to choose any \( z \)
  – if you wait for a rectangle of size \( z \) with probability \( p(z) = e^{\frac{z}{z-1}} \)
    \( \rightarrow \) randomized TCP ACK solution, which is \( e/(e-1) \) competitive
Simple TCP Congestion Scenario

- two equal senders, two receivers
- one router with infinite buffer space and with service rate $C$
- large delays when congested
- maximum achievable throughput

The TCP Congestion Control Problem

- Main Question
  How many packets per second can a sender inject into the network without overloading it?

- Assumptions
  - sender does not know the bandwidth between itself and the receiver
  - the bandwidth might change over time

- Model
  - time divided into periods $\{t\}$
  - unknown bandwidth threshold $u_t$
  - sender transmits $x_t$ packets

- Severe Cost and Gain Function
  - $\text{gain}_t = u_t - \text{cost}_t$
  - $x_t \leq u_t : \text{cost}_t = u_t - x_t \rightarrow \text{gain}_t = x_t$
  - $x_t > u_t : \text{cost}_t = u_t \rightarrow \text{gain}_t = 0$

The TCP Congestion Control Problem: The Dynamic Model

- Competitive Analysis Definition
  An online algorithm $A$ is strictly $c$-competitive if for all finite input sequences $I$
  $\text{cost}_A(I) \leq c \cdot \text{cost}_{\text{opt}}(I)$
  or
  $c \cdot \text{gain}_A(I) \geq \text{gain}_{\text{opt}}(I)$.

- The Dynamic Model
  - algorithm: chooses a sequence $\{x_t\}$
  - adversary: knows the algorithm’s sequence and chooses a sequence $\{u_t\}$

- Problem
  - Adversary is too strong: $\forall t: u_t < x_t \rightarrow \text{gain}_A = 0$

- Reasonable restrictions
  - Bandwidth in a fixed interval: $u_t \in [a, b]$
  - Multiplicatively or additively changing bandwidth from step to step
  - Changes with bursts

Bandwidth in a Fixed Interval: Deterministic Algorithm

- Preconditions
  - adversary chooses $u_t \in [a, b]$
  - algorithm is aware of the lower bound $a$ and the upper bound $b$

- Deterministic Algorithm
  - If the algorithm plays $x_t > a$ in round $t$, then the adversary plays $u_t = a$
    $\rightarrow \text{gain} = 0$
  - Therefore the algorithm must play $x_t = a$ in each round in order to have at least gain $= a$.
  - The adversary knows this, and will therefore play $u_t = b$.
  - Therefore, $\text{gain}_{\text{alg}} = a$, $\text{gain}_{\text{opt}} = b$, competitive ratio $c = b/a$. 

5/19
Bandwidth in a Fixed Interval: Randomized Algorithm

- Let’s try the ski rental trick!
  - For all possible inputs \( u \in [a, b] \) we want the same competitive ratio:
    \[ c \text{gain}_{alg}(u) = \text{gain}_{opt}(u) = u \]

- Randomized Algorithm
  - We choose \( x = a \) with probability \( p_a \) and any value in \( x \in (a, b] \) with probability density function \( p(x) \), with
  \[ p_a + \int_a^b p(x)dx = 1. \]

- Theorems
  - There is an algorithm that is \( c \)-competitive, with \( c = 1 + \ln(b/a) \).
  - There is no randomized algorithm which is better than \( c \)-competitive, with \( c = 1 + \ln(b/a) \).

- Remark
  - Upper and lower bound are tight.

Changes with Bursts

- Bursty Adversary
  - 2 parameters: \( \mu \geq 1 \) and maximum burst factor \( B \geq 1 \)
  - adversary chooses \( u_{t+1} \) from the interval
    \[ \left[ \frac{u_t}{\beta_{t+1}}, u_t \cdot \beta_{t+1} \cdot \mu \right] \]
  - where \( \beta_t = \min \{ B, \beta_{t-1} \frac{\mu}{c_{t-1}} \} \)
    is the burst factor at time \( t \) and
    where \( c_{t+1} = u_t / u_{t-1} \) if \( u_t > u_{t-1} \) and \( u_{t-1} / u_t \) otherwise

\[ u_t \]
\[ \forall t : u_{t+1} \geq u_t \]
\[ t \]

Multiplicatively Changing Bandwidth

- Preconditions
  - adversary chooses \( u_{t+1} \) such that \( u_t / \mu \leq u_{t+1} \leq \mu u_t \) with \( \mu \geq 1 \), e.g. 1.05
  - algorithm knows \( u_1 \) and \( \mu \)

- Algorithm \( A_1 \)
  - after a successful transmission in period \( t \), the algorithm chooses \( x_{t+1} = \mu x_t \)
  - otherwise: \( x_{t+1} = x_t / \mu^3 \)

- Theorem
  - The algorithm \( A_1 \) is \((\mu^4 + \mu)\)-competitive

- Algorithm \( A_2 \)
  - after a successful transmission in period \( t \), the algorithm chooses \( x_{t+1} = \mu x_t \)
  - otherwise: \( x_{t+1} = x_t / 2 \)

- Theorem
  - The algorithm \( A_2 \) is \((4\mu)\)-competitive