Optimization Problems in Congestion Control

R. Karp
karp@icsi.berkeley.edu

E. Koutsoupias
elias@cs.ucla.edu

C. Papadimitriou
christos@cs.berkeley.edu

S. Shenker
shenker@icsi.berkeley.edu

Abstract

One of the crucial elements in the Internet's success is its ability to adequately control congestion. This paper defines and solves several optimization problems related to Internet congestion control, as a step toward understanding the virtues of the TCP congestion control algorithm currently used and comparing it with alternative algorithms. We focus on regulating the rate of a single unicast flow when the bandwidth available to it is unknown and may change over time. We determine near-optimal policies when the available bandwidth is unchanging, and near-optimal competitive policies when the available bandwidth is changing in a restricted manner under the control of an adversary.

1 Introduction

The Internet carries packets on a best-effort basis with no guarantees as to when, or even if, packets will be delivered. When the Internet is congested, the resulting large packet delays and high packet drop rates seriously degrade the performance of most Internet applications. One of the crucial elements in the Internet's success has been its ability to adequately control congestion. The predominant form of congestion control is embodied in the TCP protocol. In oversimplified terms, when TCP suffers a packet loss, it decreases its sending rate (by decreasing its window size by a factor of two); when a packet is successfully delivered, it increases its sending rate (by increasing its window size by one) [2]. This process of additively increasing and multiplicatively decreasing (AIMD) the transmission rate can be thought of as a probing algorithm designed to find the maximal rate at which TCP can send packets under current conditions without incurring packet drops. While the AIMD approach is widely considered the most appropriate one —largely based on its empirical success and on certain control-theoretic arguments of Chiu and Jain [1]— here we seek to broaden our understanding of congestion probing algorithms in a more algorithmic direction. We formulate congestion probing as an optimization problem described in Section 2. We first introduce, in Section 2.1, the case when the other traffic remains constant, which turns out to be a novel and intriguing variant of binary search. But of course, the other traffic is *not* constant; in Section 2.2 we introduce the on-line algorithm problem of determining the worst-case behavior of probing strategies in the presence of changing available bandwidth. We present our technical results for the static and dynamic cases in Sections 3 and 4 respectively. We conclude in Section 5 with a brief discussion of open problems.

Before turning to our technical results, we want to emphasize that we are not attempting to accurately model what happens in the Internet and are not suggesting that TCP should be redesigned along the lines suggested by our results. Quite the contrary, we acknowledge that practical experience suggests that TCP's congestion control algorithm is better than many of the proposed alternatives. The problem is that we have as yet no formalism with which to express this superiority. In short, *while TCP may be* *the answer, we have yet to define the question.* We view this paper, in which we define, and solve, some simple optimization problems arising from congestion control, as an initial step in a research program that progressively adds additional reality to our models in an attempt to identify the question to which TCP is the answer.

2 Model

As discussed in the Introduction, one can view congestion control algorithms as a form of probing; the goal is to determine the maximal bandwidth currently available, and the result of the probe is either a successful transmission of all packets, or the dropping of one or more. TCP's congestion control algorithm increases the rate additively upon success, and decreases the rate multiplicatively upon failure. We attempt to deepen our understanding of such dynamic probing algorithms through a series of simplified models.

We consider the problem of regulating the rate of a unicast flow from host A to host B. The bandwidth available to the flow fluctuates according to the varying requirements for bandwidth of other competing flows. Host A is provided no direct information about the competing demands for bandwidth or the topology of the Internet, but does receive some limited information as to whether the flow is experiencing packet drops, and must determine its transmission rate on the basis of this information.

We assume that time is divided into successive periods, and in each period t there is a threshold u_t, representing the maximum number of packets that A can transmit to B without experiencing packet drops. In each period t A transmits some number of packets x_t and receives immediate feedback as to whether packet drops have occurred; *i.e.*, whether $x_t > u_t$. A cost function $c(x,u)$ is given, which represents the cost of transmitting x packets in a period with threshold u. In our models, the cost reflects two major components: *opportunity cost* due to sending less than the available bandwidth when $u_t > x_t$, and *retransmission delay and overhead* due to dropped packets when $x_t > u_t$. The goal of host A is to minimize the total cost incurred over all periods or, in the case of an infinite sequence of periods, the average cost per period. Since A's only feedback from period t is whether $x_t > u_t$, A does not precisely know u_t, or $c(x_t, u_t)$.

2.1 The Static Case

We assume that the fixed threshold u is a positive integer less than or equal to a known upper bound n. The problem can be viewed as a Twenty Questions game in which the goal is to determine u at minimum cost by queries of the form "Is $x > u$?". The cost of such a query is $c(x,u)$. At any step the initial data plus the results of previous queries determine an *interval of pinning* in which the threshold must lie. A probing algorithm is a rule specifying the next query as a function of the interval of pinning. Given an upper bound n on the threshold, We wish to characterize those algorithms that minimize either the worst-case cost or the expected cost under the assumption that the threshold is drawn from the uniform distribution over $\{1, 2, \cdots, n\}$.

Notice that, for an arbitrary cost function $c(x,u)$ there is a straightforward dynamic programming algorithm with running time $O(n^3)$ to minimize expected cost. This algorithm can also accommodate a discount factor, corresponding to a geometric distribution of the number of periods for which the fixed threshold is in effect. However, in this paper we focus on the asymptotic behavior of actual "uniform" algorithms, and on lower bounds, for the following two specific cost functions:

1. The *gentle cost function*, $G_\alpha(x,u)$, which is equal to $u - x$ when $x \le u$ and to $\alpha(x-u)$ when $x > u$, where α is a constant;

2. The *severe cost function*, $S(x,u)$, which is equal to $u-x$ when $x \leq u$ and to u when $x > u$.

When $x < u$ each cost function is equal to the opportunity cost of sending only x packets when u could have been sent. When $x > u$ the two cost functions take into account the cost of compensating for packet loss, under different assumptions about the protocol's behavior in the face of packet drops. The severe cost function models the case where the protocol must wait for the first dropped packet to *time out* before resuming transmission. If we take the periods to be the length of this time-out, and assume that the first loss occurs close to the beginning of the interval, then when $x_t > u$ essentially no packets are transmitted during that period and the resulting lost bandwidth can be reasonably approximated as u. The family of gentle cost functions models the case where the protocol need not wait for lost packets to time-out (e.g., the so-called *fast-retransmit* in TCP) so u packets get through to the receiver, but there is an overhead for detecting and retransmitting the $x-u$ extra packets that are dropped.

We show that for the gentle cost function there is a simple algorithm that is essentially optimal with respect to both worst-case and expected total cost. At each step the algorithm chooses a query that divides the interval of pinning into two parts whose sizes are approximately in the ratio $\sqrt{\alpha}$ to 1. The expected cost is $\frac{\sqrt{\alpha}n}{2} + O(\log n)$ and the worst-case cost is $\sqrt{\alpha}n + O(\log n)$.

For the severe cost function we have an interesting algorithm whose worst-case cost is $O(n \log\log n)$, and we prove a lower bound on worst-case cost of $\Omega(\frac{n \log\log n}{\log\log\log n})$. We have not yet determined the optimal growth rate of the expected cost. (These results are also extended to the case where no upper bound is given on the threshold u.)

2.2 The Dynamic Case

Here the threshold may vary from step to step. We assume that the sequence $\{u_t\}$ is specified by an adversary who knows our algorithm for choosing the sequence $\{x_t\}$ of probes. Obviously, we are in the realm of *competitive analysis* [4], in which the performance of an on-line algorithm for choosing $\{x_t\}$ is compared with the best among some family of off-line algorithms for choosing $\{x_t\}$. An on-line algorithm must choose x_t knowing only its sequence $x_1, x_2, \cdots, x_{t-1}$ of previous choices and the result of comparing each of these previous choices x_i to the corresponding threshold u_i. In contrast, an off-line algorithm has the benefit of hindsight; it knows the entire sequence $\{u_t\}$ beforehand. An unrestricted off-line algorithm could simply choose x_t equal to u_t for all t, incurring a total cost of zero. For this reason it seems more fruitful to study the *gain* rather than the loss. The gain function $g(x,u)$ counts the number of transmitted packets and it is $g(x,u) = u - c(x,u)$, turning the problem into a maximization problem.

Still the adversary is so powerful that it frustrates all online algorithms. To be able to discriminate between online algorithms we level the playing field by curtailing the power of the adversary to select the threshold sequence $\{u_t\}$. The question thus arises: What are meaningful ways to do so? This is a complicated problem and an interesting one in its own right; the threshold sequences depend both on the network topology and the interaction among all hosts that inject traffic into the network. The interaction among the hosts is of a game-theoretic nature (see Section 5).

A natural (and reasonably realistic) approach is to assume that the threshold does not change too drastically in a time step. We consider the case where u_{t+1} is restricted to be in an interval $I(u_t)$ that includes u_t. The adversary is allowed to choose any value u_{t+1} in the interval. Again, the on-line algorithm finds out only whether $x_t \leq u_t$, but not u_t; therefore it may not know $I(u_t)$ but a larger interval I' that contains $I(u_t)$.

We study the severe gain function for three natural problems of this kind:

- The interval $I(u_t)$ is independent of u_t, i.e., $I(u_t) = [a, b]$, for $a, b > 0$. We show (subsection 4.1) that the optimal deterministic competitive ratio is a/b and the optimal randomized competitive ratio against an oblivious adversary is $1 + \ln(a/b)$.
- The rate of the change is bounded, i.e., $I(u_t) = [u_t/\mu, \mu u_t]$, for some constant μ. In subsection 4.2 we analyze a variant of TCP and show that its competitive ratio is at most $4\mu - 2$. We also indicate that no deterministic algorithm can have a competitive ratio less than μ.
- The change of the threshold is bounded, i.e., $I(u_t) = [u_t - \alpha, u_t + \alpha]$ for some constant α. We show (subsection 4.3) that the optimal deterministic competitive ratio is between $1 + \alpha/\beta$ and $4 + \alpha/\beta$, where β is the absolute lower bound of the threshold (usually $\beta = 1$).

We conjecture that the latter two bounds can also be improved by randomization.

3 The Static Case

Let u denote the fixed but unknown threshold. At any step t an algorithm sends a number of packets x_t, learns whether $x_t \le u$ and incurs a cost $c(x_t, u)$. Eventually the algorithm determines the integer u and incurs no further cost.

At a general step in executing an algorithm A, the outcomes of previous steps have restricted u to some interval of integers $[i..j]$, which we call the *interval of pinning*. For the cost measures we consider, it is clear that the further course of the algorithm should depend only on the interval $[i..j]$ and the outcomes of subsequent steps; it should not depend on how the interval of pinning $[i..j]$ was reached. This motivates us to define an algorithm A as a function from the set of all intervals $[i..j]$ into the positive integers, subject to the restriction that $i \le A(i,j) \le j$. We shall usually assume that there is an *a priori* upper bound n on u, which is also the parameter of our asymptotic analysis; however, we occasionally discuss how our algorithms can be extended to the *unbounded case*, which involves intervals of pinning of the form $[i..\infty]$ (and in which the performance of algorithms is expressed as a function of u).

Let c be a cost function and A an algorithm. Then $C_{c,A}(i,u,j)$ denotes the cost of executing algorithm A when the initial interval of pinning is $[i..j]$ and the threshold is u; here $i \le u \le j$. The function $C_{c,A}$ is defined recursively as follows:

$C_{c,A}(i,i,i) = 0$. If $A(i,j) > u$ then $C_{c,A}(i,u,j) = c(A(i,j),u) + C_{c,A}(i,u,A(i,j)-1)$; else $C_{c,A}(i,u,j) = c(A(i,j),u) + C_{c,A}(A(i,j),u,j)$.

We define $\text{MAXCOST}_{c,A}(i,j) = \max_{u=i}^{j} C_{c,A}(i,u,j)$, and $\text{SUMCOST}_{c,A}(i,j) = \sum_{u=i}^{j} C_{c,A}(i,u,j)$. Thus $\text{MAXCOST}_{c,A}(i,j)$ is the worst-case cost of executing Algorithm A from the initial interval of pinning $[i..j]$, and $\frac{\text{SUMCOST}_{c,A}(i,j)}{j-i+1}$ is the expected cost of Algorithm A when u is drawn from the uniform distribution over $[i..j]$.

3.1 Algorithms and Upper Bounds

We consider five algorithms. The first two, BIN and TCP, are far from optimal under our cost measures. The next three, SHRINK, UNSHRINK and GOOD, are near-optimal under different measures.

- **The Algorithm BIN** is the familiar binary search: $\text{BIN}(i,j) = \lceil \frac{j-i+2}{2} \rceil$. It is easy to see that, in the severe cost model, it has $\text{MAXCOST}_{S,\text{BIN}}(1,n) = \frac{n \log_2(n)}{2} + O(n)$ and $\text{SUMCOST}_{S,\text{BIN}}(1,n) = \frac{n^2 \log_2(n)}{4} + O(n^2)$, while in the gentle cost model $\text{MAXCOST}_{G_\alpha,\text{BIN}}(1,n) = \max(1,\alpha)n + o(n)$ and $\text{SUMCOST}_{G_\alpha,\text{BIN}}(1,n) = \frac{1+\alpha}{4}n + o(n)$.

- **The Algorithm TCP.** Starting from n, it halves the probe until it becomes less than or equal to u and it then keeps increasing the lower bound on u by 1 until u is determined. It is defined by: $\text{TCP}(1,j) = \lceil j/2 \rceil$, and, for $i > 1$, $\text{TCP}(i,j) = i+1$. Under the severe cost function its MAXCOST is $\frac{n^2}{8} + O(n)$ and its SUMCOST is $\frac{n^3}{42} + O(n^2)$.

- **The Algorithm SHRINK.** The algorithm has two major phases. In the first phase we reduce the interval of pinning from $[1,n]$ to one of the form $[2^{t-1}+1, j]$ where $j \leq 2^t$. Having achieved this goal, the algorithm then proceeds to shrink the size of the interval of pinning successively down to $2^{t-2}, 2^{t-4}, 2^{t-8}, \ldots$. This is done in a particular way which guarantees that each of the $O(\log\log t)$ shrinkages incurs cost $O(2^t)$. It follows that the worst-case cost of the algorithm is $O(n \log\log n)$.

  Specifically, the first phase is defined by the following rule: if $j > 2^k$ then $\text{SHRINK}(2^{k-1}+1, j) = 2^k + 1$. The second phase is as follows: if there exists a t such that $2^{t-1}+1 \leq i < j \leq 2^t$ and m is the largest integer such that $j - i < \frac{2^t}{2^{2^m}}$ then $\text{SHRINK}(i,j) = i + \max(1, \frac{2^t}{2^{2^{m+1}}})$.

  In terms of cost, this algorithm has the property that $\text{MAXCOST}_{S,\text{SHRINK}}(1,n) = O(n \log\log n)$. Thus this algorithm has significantly better worst-case cost than TCP and BIN under the severe cost model. We shall prove later that this algorithm is near-optimal with respect to worst-case severe cost.

  It is interesting to note that, for large n, the *vast majority* of the increasing steps in SHRINK are increments by *one*, while almost all decreasing steps are substantial —just like with the TCP protocol.

- **The Unbounded Algorithm UNSHRINK.** SHRINK can be extended to an unbounded version which we call UNSHRINK, with initial interval of pinning $[1..\infty]$. It proceeds by determining an upper bound for u by repeated squaring, then performing a binary search *to determine the logarithm of* u, and finally emulating SHRINK. Its worst-case cost under the severe cost model is $O(u \log\log u)$, which is near-optimal.

- **The Family of Algorithms GOOD$_\alpha$.** At each step, this algorithm splits the interval of pinning into two parts whose sizes are in the ratio 1: $\sqrt{\alpha}$.

  $\text{GOOD}_\alpha(i,j) = i + \max(1, \lfloor \frac{j-i}{1+\sqrt{\alpha}} \rfloor)$. It can be shown that $\text{MAXCOST}_{G_\alpha,\text{GOOD}_\alpha}(1,n) = \sqrt{\alpha}n + O(\log n)$ and $\text{SUMCOST}_{G_\alpha,\text{GOOD}_\alpha}(1,n) = \frac{\sqrt{\alpha}}{2}n^2 + O(n \log n)$. We shall prove later that this family of algorithms achieves near-optimal performance for the family of cost functions G_α, under both the MAXCOST and SUMCOST criteria.

3.2 Lower Bounds

We now define the actual *complexity* of the probing problems in our model, and prove lower bounds on it. Let $\text{MAXCOST}_c(i,j) = \min_A \text{MAXCOST}_{c,A}(i,j)$ and $\text{SUMCOST}_c(i,j) = \min_A \text{SUMCOST}_{c,A}(i,j)$. Here c is a cost function and A ranges over all probing algorithms. $\text{MAXCOST}_c(i,j)$ is the intrinsic worst-case complexity of the probing problem with cost function c and initial interval of pinning $[i..j]$. $\frac{\text{SUMCOST}_c(i,j)}{j-i+1}$ is the intrinsic expected complexity (under the uniform distribution) of the same problem. In Appendix A1 we give dynamic programming algorithms for computing these quantities exactly. Here we bound them from below in terms of functions of n.

Theorem 1 $\text{SUMCOST}_{G_\alpha}(1,n) = \frac{\sqrt{\alpha}}{2}n^2 + O(n \log n)$.

Proof. (Sketch) For the gentle cost function we observe that $\text{SUMCOST}_c(i,j)$ is a function of $j-i$. If we define $T(j-i+1) = \text{SUMCOST}_c(i,j)$ we see that the function T satisfies the following recurrence: $T(n) = \min_{k=1}^{n}(\alpha\binom{k}{2} + \binom{n-k+1}{2} + T(k) + T(n-k))$ with the boundary condition $T(1) = 0$ (see Appendix A1). In a series of stages we modify this recurrence.

Conversion to continuous variables: $T_1(n) = \min_{p\in(0,1)}(\alpha\frac{pn(pn-1)}{2} + \frac{((1-p)n+1)(1-p)n}{2} + T_1(pn) + T_1((1-p)n))$ with the boundary condition $T_1(x) = x, x \in (0,1]$.

Removal of linear terms: $T_2(n) = \min_{p\in(0,1)}(\alpha\frac{p^2n^2}{2} + \frac{(1-p)^2n^2}{2} + T_2(pn) + T_2((1-p)n))$ with the boundary condition $T_2(x) = x, x \in (0,1]$.

Change in boundary conditions: $T_3(n) = \min_{p\in(0,1)}(\alpha\frac{p^2n^2}{2} + \frac{(1-p)^2n^2}{2} + T_3(pn) + T_3((1-p)n))$ with the boundary condition $T_3(x) = \frac{\sqrt{\alpha}}{2}x^2, x \in (0,1]$

The unique solution of the recurrence for T_3 is: $T_3(n) = \frac{\sqrt{\alpha}}{2}n^2$, obtained by setting p to $\frac{1}{1+\sqrt{\alpha}}$. The proof is completed by proving the following inequalities: $| T_2(n) - T_3(n) | = O(n)$; $| T_1(n) - T_2(n) | = O(n\log n)$; $| T(n) - T_1(n) | = O(n)$. ∎

Theorem 2 $\text{MAXCOST}_{G_\alpha}(1,n) = n\sqrt{\alpha} + O(\log n)$.

Proof. (Sketch) We first relax the definition of 'algorithm' by allowing probes with fractional values. With this generalization an interval of pinning is a half-open set $(x,y]$ where neither x nor y need be an integer. The initial interval of pinning is $(0,n]$, and the algorithm terminates when the length of the interval of pinning is less than or equal to 1. It can be shown that the amount by which this relaxation reduces the optimal worst-case cost is $O(\log n)$. The worst-case cost of such an algorithm, given an initial interval of pinning $(x,y]$, depends only on the length of the interval. Let $T(z)$ be the worst-case cost starting from an interval of length z. Then $T(z) = 0$ when $z \leq 1$. Replacing this boundary condition by the artificial boundary condition $T(z) = \sqrt{\alpha}z$, $z \leq 1$, increases the worst-case cost by at most $\sqrt{\alpha}$. We then show that, using the modified boundary condition, $T(z)$ is a nondecreasing continuous function. It then follows that, when the interval of pinning is of length z, an optimal step is to select a probe that partitions the interval into parts of size pz and $(1-p)z$, such that $\alpha pz + T(pz) = (1-p)z + T((1-p)z)$. It follows that $T(z) = \alpha pz + T(pz) = ((1-p)z + T((1-p)z)$. We can then show that an optimal choice of p to minimize $T(z)$ is, for all z, $\frac{1}{1+\sqrt{\alpha}}$, yielding $T(z) = \sqrt{\alpha}z$ for all z. It follows that $\text{MAXCOST}_{G_\alpha}(1,n) = n\sqrt{\alpha} + O(\log n)$. ∎

The above proof shows that, for a continuous approximation to the problem of minimizing worst-case cost for the interval of pinning $[1..n]$ under the gentle cost function G_α, each interval of pinning should be divided into two parts in the ratio 1: $\sqrt{\alpha}$. The algorithm GOOD that was described earlier is a discrete approximation to this policy, but improves upon it by exploiting the restriction to integer values of u. It follows that $\text{MAXCOST}_{G_\alpha,\text{GOOD}}(1,n) = n\sqrt{\alpha} + O(\log n)$, and is thus worst-case optimal to within an additive term $O(\log n)$.

We next turn to the severe cost model:

Theorem 3 $\text{MAXCOST}_S(1,n) = \Omega(\frac{n\log\log n}{\log\log\log n})$.

In preparation for the proof we require a lemma about n-leaf rooted oriented binary trees. Every non-leaf node of such a tree T has a left child and a right child. Every edge is directed from a parent to one of its children. Define the weight $W(v)$ of node v as the number of leaves in the subtree rooted at v. Define the *right cost* of T as the sum, over all right children v in T, of $\binom{W(v)}{2}$. Define the *left height* of node v as the maximum, over all paths from v to a leaf, of the number of left children in the path, excluding v itself. Define the left height of T as the left height of its root. At the heart of the proof of Theorem 3 is the observation that a tree with small right cost must have large left height.

Lemma 4 *Let $g(n)$ be a function from positive integers to positive integers. Every n-leaf binary tree with right cost less than or equal to $n^2g(n)$ has left height greater than or equal to $\log_3(\frac{\ln(ng(n))}{\ln(4\sqrt{3}g(n))})$.*

Proof. Let T be a n-leaf binary tree with right cost less than or equal to $n^2g(n)$. For each k, let n_k be the maximum weight of a right node in T of left height less than or equal to k. Let v be a node of left height less than or equal to k and weight n_k. Consider the chain of right children descending from v. The left child of each node in this chain is of left height less than or equal to $k-1$, and hence of weight less than or equal to n_{k-1}. It follows that, for each i, the ith node in the chain of right children has weight greater than or equal to $n_k - in_{k-1}$, and hence contributes at least $\binom{n_k-in_{k-1}}{2}$ to the right cost of T. Hence the total contribution of the nodes in this chain to the right cost of T is at least $\sum_{i=1}^{\lfloor \frac{n_k}{n_{k-1}} \rfloor} \binom{n_k-in_{k-1}}{2}$. This sum is required to be less than or equal to $n^2g(n)$, but it is is greater than or equal to $\frac{\binom{n_k-n_{k-1}+1}{3}}{n_{k-1}}$, which in turn is greater than or equal to $\frac{(n_k-n_{k-1}-1)^3}{6n_{k-1}}$. Thus we arrive at the inequality $(n_k-n_{k-1}-1)^3 \leq 6n_{k-1}n^2g(n)$, which implies that $n_k \leq n_{k-1}+1+(6n_{k-1}n^2g(n))^{1/3}$. Since $n_{k-1} \leq n$ it follows that $n_{k-1}+1 \leq (6n_{k-1}n^2g(n))^{1/3}$. Hence $n_k \leq 2(6n_{k-1}n^2g(n))^{1/3}$. Since $n_0 = 1$ it follows by induction that, for all k, $n_k \leq 4\sqrt{3}(ng(n))^{1-\frac{1}{3^k}}$. Let t be the left height of the tree T. Then $n_t = n$. Hence $n \leq 4\sqrt{3}(ng(n))^{1-\frac{1}{3^t}}$, from which it follows that $t \geq \log_3(\frac{\ln(ng(n))}{\ln(4\sqrt{3}g(n))})$. ∎

Corollary 5 *There is a positive constant D such that, for all n, and for every n-leaf binary tree T, either $leftheight(T) \geq D\frac{\ln\ln(n)}{\ln\ln\ln(n)}$ or $rightcost(T) \geq Dn^2\frac{\ln\ln(n)}{\ln\ln\ln(n)}$.*

For the proof of Theorem 3, note that, for any given initial interval of pinning, a probing algorithm A can be represented as a rooted binary tree. The nodes of the tree are the intervals of pinning that can occur in the course of the algorithm. The root is the initial interval of pinning and the leaves are intervals of the form $[i..i]$. Node $[i..j]$, where $j \neq i$, has the left child $[i..A(i,j)]$ and the right child $[A(i,j)+1..j]$.

Proof. (Of the Theorem.) Clearly $\text{MAXCOST}_S(1,2n) \geq \text{MAXCOST}_S(n+1,2n)$. Consider any probing algorithm A. The execution of A with initial interval of pinning [n+1, 2n] can be represented by a n-leaf rooted, oriented binary tree T. It is easily verified from the definition of the severe losss function S that $\text{MAXCOST}_{S,A}(n+1,2n) \geq (n+1)leftheight(T)$ and $\text{SUMCOST}_{S,A}(n+1,2n) \geq rightcost(T)$. It follows that $\text{MAXCOST}_{S,A}(n+1,2n) \geq \frac{rightcost(T)}{n}$. Applying the above corollary we obtain $\text{MAXCOST}_{S,A}(n+1,2n) \geq D\frac{\ln\ln(n)}{\ln\ln\ln(n)}$. Since A was an arbitrary probing algorithm it follows that $\text{MAXCOST}_S(1,n) = \Omega(\frac{\ln\ln(n)}{\ln\ln\ln(n)})$. ∎

There is some looseness in the proof of this lower bound. We conjecture that $\text{MAXCOST}_S(1,n) = \Omega(n\log\log n)$, matching the worst-case complexity of the algorithm SHRINK given earlier.

4 The Dynamic Case

Consider an online probing algorithm with probe sequence $\{x_t\}$. Its gain at time t is $g(x_t,u_t) = x_t$ when $x_t \leq u_t$ and $g(x_t,u_t) = 0$ when $x_t > u_t$. This definition of gain corresponds to the severe cost function. The total online gain is therefore $\text{gain}_n = \sum_{t=1}^{n} g(x_t, u_t)$ while the optimal (offline) gain is $\text{opt}_n = \sum_{t=1}^{n} u_t$. The online algorithm has competitive ratio r if

$$r\,\text{gain}_n \geq \text{opt}_n + \text{const},$$

where const depends only on the initial conditions.

From the competitive analysis point of view the study of gain is much more meaningful than the study of loss (cost). It is definitely more informative—in all cases studied here the competitive ratio with respect to loss turns out to be trivial, either 1 or ∞.

It is easy to see that if the adversary is allowed to select any sequence of thresholds $\{u_t\}$, then there exists no competitive online algorithm. In this section we consider three natural ways to restrict the power of the adversary.

4.1 Adversary restricted to a fixed interval

We first consider the simple case when the adversary can choose any threshold from a fixed interval, i.e., $u_t \in [a, b]$. The deterministic case is completely trivial: An optimal online algorithm would never select a rate $x_t > a$ because of adversary's threat to select $u_t = a$. Thus the optimal online algorithm transmits at the minimum rate $x_t = a$. But in that case the adversary will select the maximum possible bandwidth $u_t = b$ yielding a competitive ratio of b/a.

For the randomized case, the situation is more interesting, as randomization improves the competitive ratio exponentially. We consider *oblivious* adversaries [3], that is, adversaries that have to select the whole sequence $\{u_t\}$ in advance (unrelated to the random choices of the online algoritm).

Theorem 6 *The optimal randomized competitive ratio against an adversary that is constrained to select $u_t \in [a, b]$ is $1 + \ln(b/a)$.*

Proof. We consider a memoryless randomized algorithm. At every step the algorithm selects x_t according to the following probability density function: $f(x) = \frac{1}{rx}$ where is $r = 1 + \ln(b/a)$ for all $x > a$. The case $x_t = a$ is treated in a special way: a is selected with probability $\frac{1}{r}$ (it is easy to check that the probabilities sum to 1). When the adversary selects $u_t = y$, the online gain is equal to $a\frac{1}{r} + \int_a^y f(x)x\,dx = y/r$. The optimal gain is y and the competitive ratio is r, independently of the choice of y.

To show that this is optimal we employ Yao's Lemma [5] (the classical minimax theorem of Game Theory adapted to on-line algorithms): It suffices to consider a randomized adversary against deterministic on-line algorithms. In particular, let the adversary select y with probability density function $g(y) = a/y^2$; in a similar manner with the upper bound, the remaining probability $\frac{a}{b}$ is assigned to b. If the online algorithm selects $x_t = x$, its gain is $x\int_x^b g(y)\,dy + x\frac{a}{b} = a$. The expected optimal cost is $\int_a^b g(y)y\,dy + b\frac{a}{b} = a(1 + \ln(b/a))$ and the ratio is $1 + \ln(b/a)$, independently of the online choice x. ∎

In reality both the thresholds $\{u_t\}$ and the online rates $\{x_t\}$ take integer values, while the results of this section hold for real values; this does not affect the analysis in any significant way (for example, for integer values the ratio is not $1 + \ln(b/a)$ but $1 + H_b - H_a$ where $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n$).

4.2 Adversary restricted by a multiplicative factor

We now consider the case when the adversary can change the threshold by a multiplicative factor. In particular, we assume that the adversary can select any threshold u_{t+1} in the interval $[u_t/\mu, \mu u_t]$ for some constant $\mu \geq 1$. We show that a variant of TCP achieves competitive ratio at most $4\mu - 2$; this is optimal within a factor of 4 since no deterministic online algorithm can have a competitive ratio less than μ.

An interesting observation is that the the restriction $u_t/\mu \leq u_{t+1}$ is useless to our online algorithm. We show a stronger result by allowing the adversary to decrease the threshold u_{t+1} arbitrarily. This doesn't affect the competitive ratio. The underlying reason is that the extra power of the adversary buys it nothing in the face of a controlled (i.e., multiplicative) online decrease—the optimal adversarial policy is to choose a threshold either slightly less than the online rate or much greater than it.

Theorem 7 *There is a deterministic online algorithm with competitive ratio* $(\sqrt{\mu} + \sqrt{\mu - 1})^2$ *against an adversary who is constrained to select any threshold* u_{t+1} *in the range* $[0, \mu u_t]$*, for some constant* $\mu \geq 1$*. On the other hand, no deterministic online algorithm can achieve a competitive ratio better than* μ*.*

Proof. We will analyze the following algorithm:

> After a successful transmission the online algorithm raises its rate to $x_{t+1} = \mu x_t$. After a failed transmission it lowers its rate (also by a fixed factor) to $x_{t+1} = \lambda x_t$.

In our analysis we will use the (optimal) decreasing factor $\lambda = \frac{\sqrt{\mu}}{\sqrt{\mu}+\sqrt{\mu-1}}$ which results in competitive ratio $r = (\sqrt{\mu} + \sqrt{\mu - 1})^2$. Interestingly this value of λ is approximately $1/2$ (for large μ). Indeed, the algorithm that uses $\lambda = 1/2$ (a value independent of μ) has competitive ratio $4\mu - 2$, not much worse than the optimized ratio.

The following two invariants are maintained (see Appendix A2 for the simple inductive proof):

- $u_t \leq \frac{\mu}{\lambda} x_t$, and
- $r\,\text{gain}_t \geq \text{opt}_t + \Phi(x_{t+1}) - \Phi(x_1)$, where $\Phi(x) = \frac{1}{1-\lambda} x$ is an appropriate *potential function.*

The theorem follows from the second invariant.

The lower bound is simple: the adversary can select as u_t, for all $t \geq 1$, any value in the interval $[u_1, \mu u_1]$. This is consistent with the constraint $u_{t+1} \leq \mu u_t$. The lower bound follows (this is the case of subsection 4.1). ∎

The upper bound $(\sqrt{\mu} + \sqrt{\mu - 1})^2 < 4\mu - 2$ is within a factor of 4 of the lower bound. It remains an open problem to close the gap. Notice also that both upper and lower bounds hold even with the restriction $u_{t+1} \geq u_t/\mu$.

4.3 Adversary restricted by an additive term

We now turn our attention to adversaries that can change the bandwidth by a fixed integer amount α. More precisely, we assume that the adversary is constrained not to change the threshold by a constant α, i.e., $u_{t+1} \in [u_t - \alpha, u_t + \alpha]$. As in the case of multiplicative increase, the restriction $u_{t+1} \geq u_t - \alpha$ doesn't seem to help the online algorithm. In contrast, if we allow the threshold to be arbitrarily small, there is no algorithm with bounded competitive ratio. For example, in that case the adversary can select any u_t in the interval $[0, \alpha]$ and it follows from the results of subsection 4.1 that the competitive ratio (deterministic and randomized) is infinite. Fortunately, in real life the threshold has the natural lower bound of 1 bit or packet (we can ignore the time steps when the bandwidth is 0 since neither the online nor the offline algorithm transmit anything). Therefore we shall further assume that the threshold u_t has a constant lower bound β.

Theorem 8 *The optimal deterministic competitive ratio against an adversary constrained to select threshold u_{t+1} in the interval $[\beta, u_t + \alpha]$ is at most $4 + \alpha/\beta$. On the other hand, no deterministic online algorithm has competitive ratio better than $1 + \alpha/\beta$.*

Proof. We consider the following class of natural on-line algorithms.

> After a successful transmission the online algorithm raises its rate to $x_{t+1} = x_t + \gamma_1$. After a failed transmission it lowers its rate to $x_{t+1} = x_t - \gamma_2$ (or to β, if $x_t - \gamma_2 < \beta$).

One can compute the parameters γ_1 and γ_2 that minimize the competitive ratio but the analysis is complicated. To avoid this complication in this abstract, we restrict our attention to online algorithms that have $\gamma_1 = \gamma_2 = \gamma$; it turns out that the competitive ratio does not deteriorate much. We use an appropriate value $\gamma = \alpha/(r-1)$ where r is the competitive ratio (approximately $4 + \alpha/\beta$).

The proof has the same flavor as the proof of Theorem 7. It is not hard to show that the following two invariants are maintained.

- $u_t \le (r-1)x_t + \gamma$ and
- $r\text{gain}_t \le \text{opt}_t + \Phi(x_{t+1}) - \Phi(x_1)$,

where $\Phi(x) = \frac{x^2}{2\gamma} + \frac{x}{2}$ is a potential function. However, the following technical (but inessential) assumption is needed here: The initial rate x_1 (and consequently every subsequent online rate) is of the form $\beta + m\gamma$ for some integer m. In fact, a more careful accounting shows that the above invariants yield a slightly better competitive ratio: $r = \frac{z+3+\sqrt{z^2+10z+1}}{2}$, where $z = \alpha/\beta$.

The lower bound follows from the observation that the adversary can always select any threshold u_t in the range $[\beta, \beta + \alpha]$. ∎

5 Open Problems

We mention a few open problems suggested by our results. Most obviously, there is a $\log\log\log n$ gap between the performance of SHRINK and the lower bound; we conjecture that our lower bound proof can be tightened but this remains to be verified. Moreover, we do not yet have a lower bound for the average cost of the severe cost function.

An important future research direction is to enrich our model with *game-theoretic* features. The available bandwidth, which in our current treatment is either constant (in Section 3) or chosen by an adversary (in Section 4), is in fact the result of other flows probing for their own available bandwidth. One can view this set of probing flows as playing a game against each other where each player only receives limited feedback from their choice of strategy; the natural question is, what is the Nash equilibrium of this game?

References

[1] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. In *Journal of Computer Networks and ISDN*, 17(1):1-14(1989).

[2] V. Jacobson. Congestion Avoidance and Control In *ACM SigComm Proceedings*, pp 314-329, 1988.

[3] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[4] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] A. C. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Symp. Foundations of Computer Science*, pages 222–227, 1977.

A: Appendix

A1. Optimality

We define: $\text{MAXCOST}_c(i,j) = \min_A \text{MAXCOST}_{c,A}(i,j)$ and $\text{SUMCOST}_c(i,j) = \min_A \text{SUMCOST}_{c,A}(i,j)$. Here c is a cost function and A ranges over all probing algorithms. $\text{MAXCOST}_c(i,j)$ is the intrinsic worst-case complexity of the probing problem with cost function c and initial interval of pinning $[i..j]$. $\frac{\text{SUMCOST}_c(i,j)}{j-i+1}$ is the intrinsic expected complexity (under the uniform distribution) of the same problem.

By giving efficient dynamic programming algorithms we show that $\text{SUMCOST}_c(i,j)$ can be computed easily for any cost function, and $\text{MAXCOST}_c(i,j)$ can be computed easily for the cost functions S and G_α.

For brevity let $F(i,j)$ denote $\text{SUMCOST}_c(i,j)$, where c is a given cost function. We obtain the following recurrence, which allows $F(i,j)$ to be computed in $O((j-i+1)^3)$ steps, where a step is an addition, subtraction, comparison or evaluation of the cost function.)

$$F(i,j) = \min_{k=i}^{j-1}(F(i,k) + F(k+1,j) + \sum_{u=i}^{j} c(k,u))$$

$$F(i,i) = 0$$

In particular $F(i,n)$ can be calculated in $O(n^3)$ steps. Using simple modifications of the algorithm, It is possible to incorporate a finite time horizon or a discount factor, to model the possibility that the threshold will remain constant for only a limited time,

In the case of the severe cost function S this recurrence simplifies as follows:

$$F(i,j) = \min_{k=i}^{j-1}(F(i,k) + F(k+1,j) + \binom{k}{2} - \binom{i}{2} + \binom{j-k+1}{2}$$

$$F(i,i) = 0$$

In the case of the gentle cost functions one obtains a simplification by noting that $F(i,j)$ is a function of $j-i$. Introducing the function T such that $T(k)$ is the common value of $F(i,j)$ for all intervals such that $j-i+1=k$, we obtain the recurrence

$$T(n) = \min_{k=1}^{n-1}(\alpha\binom{k}{2} + \binom{n-k+1}{2} + T(k) + T(n-k))$$

with the boundary condition $T(1) = 0$. This recurrence can be solved in time $O(n^2)$.

For each of the severe and gentle cost functions we conjecture that there is an algorithm A which minimizes SUMCOST for every initial interval of pinning and has the following monotonicity property: $A(i,j) \leq A(i,j+1)$. If this is true we can reduce the time bounds for the algorithms to compute $\text{SUMCOST}_S(1,n)$ and $\text{SUMCOST}_{G_\alpha}(1,n)$ to $O(n^2)$ and $O(n)$, respectively.

The dynamic programming algorithms to compute $\text{MAXCOST}_S(i,j)$ and $\text{MAXCOST}_{G_\alpha}(i,j)$ require a further trick involving an extension of the function F. For any real number r and any severe or gentle cost function c define $F(r,i,j) = \min_A max_{u=i}^{j}[ru + COST_{c,A}(i,u,j)]$, where A ranges over all algorithms. Note that $F(i,j) = F(0,i,j)$.

For the severe cost function S we obtain: $F(r,i,j) = \min_{k=i}^{j-1}(\max(F(r+1,i.k), F(r+1,k+1,j)-k))$, with the boundary condition $F(r,i,i) = 0$ for all r and $F(n,i,j) = \infty$. Here r ranges over integers between 1 and n. This yields an algorithm to compute $\text{MAXCOST}_S(1,n)$ in time $O(n^4)$. If we can obtain an upper bound h on the number of steps in an optimal probing algorithm with initial interval of pinning $[1..n]$ then we only need consider integers r in the range from 1 to $h+1$ and we obtain the time bound $O(n^3 h)$. It should be possible to obtain a bound on h of order *polylog*(n), but we have not done so.

For the gentle cost functions G_α we obtain: $F(r,i,j) = \min_k(\max(\alpha k + F(r-\alpha,i,k), F(r+1,k+1,k)-k))$ with the boundary conditions $F(r,i,i) = ri$ and $F(r,i,j) = \infty$ unless r is of the form $a - \alpha b$ where a and b are nonnnegative integers summing to at most n. Thus, if α is the ratio of integers A and B, then only $n(A+B)$ different values of r need be considered, and the running time of the algorithm to compute $\text{MAXCOST}_{G_\alpha}(1,n)$ runs in time $O(n^4(A+B))$.

A2: Full Proof of Theorem 7

We will analyze the following algorithm:

> After a successful transmission the online algorithm raises its rate to $x_{t+1} = \mu x_t$. After a failed transmission it lowers its rate (also by a fixed factor) to $x_{t+1} = \lambda x_t$.

In our analysis we will use the (optimal) decreasing factor $\lambda = \frac{\sqrt{\mu}}{\sqrt{\mu}+\sqrt{\mu-1}}$ which results in competitive ratio $r = (\sqrt{\mu}+\sqrt{\mu-1})^2$. Interestingly this value of λ is approximately $1/2$ (for large μ). Indeed, the algorithm that uses $\lambda = 1/2$ (a value independent of μ) has competitive ratio $4\mu - 2$, not much worse than the optimized ratio.

We will argue that the following two invariants are maintained:

- $u_t \le \frac{\mu}{\lambda} x_t$, and
- $r\,\text{gain}_t \ge \text{opt}_t + \Phi(x_{t+1}) - \Phi(x_1)$, where $\Phi(x) = \frac{1}{1-\lambda}x$ is an appropriate *potential function*.

The theorem follows from the second invariant. We show the two invariants by induction. The proof is straightforward (the "hard" part was to come up with the right potential function Φ). The base case is trivial (without loss of generality we assume that the online algorithm knows the initial threshold).

We consider first the case when the online algorithm succeeds at time t. The online gain is $\Delta\text{gain} = x_t$; also the next online rate is raised to $x_{t+1} = \mu x_t$. It is obvious that the first invariant is maintained because $u_{t+1} \le \mu u_t$. To check the second invariant we observe that the optimal gain increases by $\Delta\text{opt} = u_t \le \frac{\mu}{\lambda} x_t$. The second invariant follows from the inequality $r\,\Delta\text{gain} \ge \Delta\text{opt} + \Phi(x_{t+1}) - \Phi(x_t)$, which can be verified straightforwardly.

The case of failure is similar. We first observe that we must have $u_t < x_t$. The online gain now is 0 and the next rate decreases to $x_{t+1} = \lambda x_t$. The next threshold u_{t+1} is at most $\mu u_t < \mu x_t$. It is again straightforward to check that both invariants are maintained.