

Cloud Control

Voluntary Admission Control for Intranet Traffic Management

John Langford · Lihong Li · Preston McAfee · Kishore Papineni

Received: date / Accepted: date

Abstract The Internet backbone of many corporations carries two kinds of traffic: urgent and delayable. Shifting some traffic from peak periods to valleys reduces capacity requirements. We consider the case of managing the delayable traffic by an admission control (AC) system. AC gets link utilization feedback every τ seconds. Delayable opt-in sources obtain permission from AC to transmit for up to τ seconds at a rate not exceeding a limit imposed by AC, renewing permission as needed. Urgent traffic bypasses AC. AC must allocate bandwidth to competing delayable traffic sources. We prove that among all throttling transformations of flows that achieve a desired mean aggregate flow, rate limits on flows minimize the variance of their sum. Furthermore, a single rate limit common to all flows achieves the optimum. Thus, for a single link, AC must decide on a single rate limit for all delayable sources in each τ -second cycle. We evaluate different policies that set the rate limit dynamically in an empirical setting using netflow records on a link on the backbone of Yahoo!. Using historical data, we also derive the best possible reduction in capacity of this link using a closed-form solution to an assignment problem. We show that AC achieves capacity reduction close to the best possible reduction.

J. Langford
Yahoo! Research, New York
Tel.: 1-212-571-8151
Fax: 1-212-571-4490
E-mail: jl@yahoo-inc.com

L. Li
Yahoo! Research, Santa Clara, CA
Tel.: 1-408-349-4299
E-mail: lihong@yahoo-inc.com

P. McAfee
Yahoo! Research, Burbank, CA
Tel.: 1-818-524-3290
E-mail: mcafee@yahoo-inc.com

K. Papineni
Yahoo! Research, New York
Tel.: 1-212-571-8147
E-mail: kpapi@yahoo-inc.com

Keywords admission control · rate throttling

1 Introduction

Many large corporations have a wide-area network that carries two kinds of traffic: urgent traffic and traffic that can be delayed by a few minutes to a few hours. There is a big potential for automatic traffic shaping on these corporate networks. We assume that some producers of delayable traffic can be incented to opt into a scheme of rate-throttling by an external agent. We assume that we can trap calls to TCP/UDP at the operating system level on the source and that we can rewrite the called methods to communicate with an admission control (AC) server. While we assume that application software that generates delayable traffic does not itself change, we do assume that it can be relinked with a new library. In contrast, sources of urgent traffic or receivers of any type of traffic need not even be aware of the admission control mechanism.

We focus on a single link. A variety of sources of traffic use this link. Our mechanism involves an AC that receives periodic feedback about traffic flow from the link. Sources can ignore AC (such sources are called class-1); those that choose or are compelled to use AC (such sources are called class-2) ping AC periodically for permission to inject traffic into the link. The problem we address is the design of AC's instructions, as well as the needed capacity on the link.

1.1 Prior work

Several authors [1–3, 5, 6] have attacked a similar problem with different approaches. For example, TCP-LP [3] and TCP Nice [6], are protocol alterations for background transfer support. The approach outlined here is implementable with simple user-land sender-side trapping of appropriate library calls, and hence is substantially less invasive of existing systems and easier to implement. We also run simulations fed by *real* data rather than simply simulated network flows, both validating the need for background transport control generally, as well as providing some reasonable design parameters. In these papers, fairness is assumed to be a desired criterion while we show that fairness *is* a desired criterion, as it reduces variance allowing improved network utilization.

In [1] the authors present an application level receiver-side control mechanism, while the approach we consider here is an application level sender-side control mechanism. Our experiments with real data suggest that very tight control delays on the order of 10 seconds or less are required in order to effectively deal with real-world demand burstiness. This timescale is substantially shorter than the control timescales demonstrated in existing *ns* experiments for this approach, as well as TCP-LP.

In [2] the authors propose an improvement, HARP, on TCP by installing gateways and relays, which can be used as an overlay to a TCP system. HARP uses real-time delay feedback to re-route packets and optimize the network. Only non-urgent traffic is delayed, as recognized by the gateways. Substantial speed gains are achieved in simulations. One downside is the installation of additional network hardware and the need to reassemble and reorder packets, which no longer arrive in order.

1.2 Outline

The rest of the paper is organized as follows. In Section 2, we derive a closed-form solution to optimal retrospective traffic shifting, given historical traces of different classes of traffic. In Section 3, we consider the case of a single link with multiple sources of two classes of traffic (urgent and delayable). We show that even when we know in advance the probability density functions of flow rates of the delayable sources, a single rate limit uniformly applied to them minimizes variance. This considerably simplifies the solution: we just need to determine the rate limit dynamically (every τ seconds) based on future projections of urgent traffic, network status, past requests, and the target capacity utilization. We describe the AC formalism and an evaluation metric in Section 4. In Section 5, we present a simple baseline rate limit algorithm, fashioned after the AIMD (additive increase multiplicative decrease) scheme used in TCP. Section 6 describes a machine learning approach to admission control. Section 7 presents experimental results.

2 Optimal Retrospective Traffic Shifting

Before building a traffic shaping mechanism, we would like to know upfront whether the effort is justified. For traffic shaping to work, we first need to be able identify different classes of traffic. We collected traffic traces on links of the Yahoo! backbone. A traffic flow trace contains, among other things, the source IP address, source port, destination IP address, destination port, router IP address, input interface ID, output interface ID, flow start and end times in milliseconds since a reference time, transport protocol (UDP or TCP), and some flags. At Yahoo, there is a way to identify data archival and business-continuity-related file copies from one data center to another. Such traffic always originates on specially designated ports. Based on the source port numbers, we classified traffic into two classes: class-1 (urgent), class-2 (some delay tolerable). Our classification is not comprehensive in that we may have missed many sources of class-2 traffic.

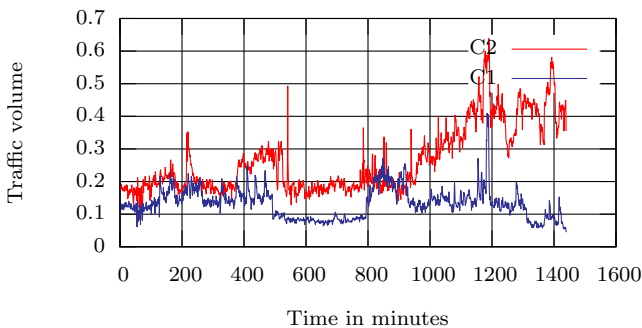


Fig. 1 1-day traffic

Figure 1 shows aggregated traffic in these classes on a certain link on the firm's backbone.¹ Since capacity planning is link-specific anyway, we will focus on a single link in the rest of the paper. It is noteworthy that class-1 traffic is nonuniform over time and offers potential to reduce bandwidth capacity. Reduce by how much? The proposition below helps calculate the potential reduction. First, some notation. Let time be discrete, from 1 to T . Let $X(t, s)$ be the traffic that originates at time t and must be transported by time s . We seek the minimum capacity γ^* that handles all transport:

Proposition 1 *Minimum required capacity is*

$$\gamma^* = \max_{1 \leq t \leq s \leq T} \frac{1}{s-t+1} \sum_{r=t}^s X(r, s)$$

Proof. Note that $\sum_{r=t}^s X(r, s)$ is the total traffic that must be transported during $[t, s]$ which consists of $s-t+1$ periods. Therefore, $\sum_{r=t}^s X(r, s) \leq \gamma^*(s-t+1)$ for all t, s . By definition, γ^* should be the smallest number that satisfies these inequalities. Therefore it equals the maximum of these lower bounds. ■

As an example of using the above proposition, consider Figure 1. Urgent (C1) traffic comes under $X(t, t)$ since such traffic cannot be delayed. We can assume a maximum permissible delay d for C2 and calculate the minimum capacity required to clear all traffic under these conditions. C2 traffic originating at time t must be cleared during $[t, t+d]$, hence becomes part of $X(t, t+d)$. Figure 2 shows the gain in capacity as a function of maximum admissible delay in class-2. Gain is $100 \frac{\gamma_0 - \gamma^*}{\gamma_0}$ where γ_0 is the minimum capacity needed if no delay is allowed.

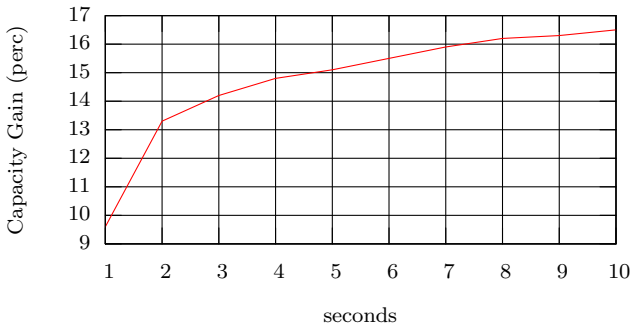


Fig. 2 Retrospective capacity gain

3 Rate Control of Many Sources

It is clear from the previous section that, even with a possible underestimate of class-2 traffic, we can benefit substantially from delaying class-2 traffic by a few seconds. Let

¹ There is much variability in volume and class-ratios across days.

c be the target capacity of the link (perhaps set to, say, 90% of the true installed capacity). The goal of admission control is to inject class-2 traffic into the link such that class-1 and class-2 traffic together never exceeds c , but stays close to it as much as possible. In other words, AC must allocate bandwidth to individual sources of class-2 traffic in such a way that class-2 throughput is maximized while the link is not overloaded. How should we allocate bandwidth to class-2 sources? It is clear that we would like the aggregate flows to be as uniform as possible, other things being equal. We model the desire for steady traffic by minimizing the variance, subject to a mean flow of traffic, which may equal the available head room. For the moment, assume that we know the head room h exactly and that there are n controllable sources (of class-2 traffic). Source i generates flow X_i , a random variable. Suppose we know $f(\mathbf{x})$, the joint probability density function of $\mathbf{X} = (X_1, X_2, \dots, X_n)$. Given this extensive knowledge, could we design a rate-throttling transformation specifically for each source in order to reduce variance in the aggregate flow? In principle, the throttling transformation on a particular source could depend on the flow rates at other sources. But it is impractical to measure the flow rates for all sources to adjust flow rate of an individual source. Thus we assume that throttling transformation of an individual source depends only on its own flow. Formally, a throttling transformation is a (Borel-measurable) function t that reduces the flow rate: $0 \leq t(x) \leq x$. What is the optimal form of $t_i(\cdot)$? Is it continuous? How does it depend on f ? The optimization problem is formalized as

$$\inf_{t_i(\cdot)} \sum_i \mathbf{E}(t_i(X_i) - \mathbf{E}(t_i(X_i)))^2$$

subject to

$$\begin{aligned} 0 &\leq t_i(x) \leq x \\ \sum_i \mathbf{E}t_i(X_i) &= h \end{aligned}$$

The answer turns out to be exceedingly simple:

Theorem 1 *For all throttling transformations, t_i on flows with a mean aggregate flow of $h = \sum_i \mathbf{E}t_i(X_i)$, there exists a rate limit r , such that setting $t_i(X) = X$ for $X \leq r$ and $t_i(X) = r$ otherwise minimizes the variance of the sum of flows.*

Proof. Let $f_i(\cdot)$ be the marginal probability density of traffic at source i . Because $t_i(\mathbf{X}) = t_i(X_i)$ and because of the constraint on the aggregate mean, the optimization problem is equivalent to minimizing

$$\frac{1}{2} \sum_i \int_0^\infty t_i^2(x) f_i(x) dx$$

subject to the constraints. We will ignore the nonnegativity constraint for simplicity. The Lagrangian is

$$\begin{aligned} &\frac{1}{2} \sum_i \int_0^\infty t_i^2(x) f_i(x) dx + \\ &\sum_i \int_0^\infty \mu_i(x) (t_i(x) - x) f_i(x) dx + \\ &\lambda (h - \sum_i \int_0^\infty t_i(x) f_i(x) dx) \end{aligned}$$

By Euler-Lagrange, we have

$$t_i(x) + \mu_i(x) - \lambda = 0$$

where $\mu_i(x) \geq 0$ with equality whenever $t_i(x) < x$. Therefore either $t_i(x) = x$ or $\lambda = t_i(x) < x$. In other words, $t_i(x) = \min\{x, \lambda\}$. ■

Remarkably, the throttling transformation on each source is simply a rate cap that is common to all sources. Note that we did not assume that the sources are independent. Were we to know $f_i(x)$, we would solve the following for r :

$$h = nr - \sum_i \int_0^r \int_0^x f_i(t) dt dx$$

In practice, we would not know $f_i(\cdot)$ a priori. Whereas one could attempt to construct estimates of $f_i(\cdot)$ for each source online, this theorem suggests that we can simply adjust the rate limit online based on feedback from the link. In the next section, we describe the experimental setup based on actual netflow records of a link on Yahoo's backbone.

4 AC Formalism and Evaluation

We assume that feedback from the network is in the form of link utilization, centrally collected at AC directly from network devices periodically, once every τ seconds. The link feedback is in the form of total bytes transferred since last polled, and does not contain class-level information. In our setup, sources request AC for permission to send bits. Given that we trap calls below the application layer, this permission request does not include the total number of bits to be transported. The request contains the IP address and port of the source as well as the destination. The AC grants permission in the form of a short-term (up to τ seconds) nonexclusive lease of bandwidth with a rate limit. The source can send flows not exceeding the rate limit for the duration of the lease and must return to AC for a new lease to continue transport as necessary. A rate limit of zero means pure delay. In this setup, control overhead occurs roughly every τ seconds – in the form of polling the devices and communication between AC and sources. Traffic is throttled at the sources, not when in transit. AC does not know the round-trip times from sources to destinations. The only information available to AC is the link utilization status from routers and the past requests from sources, and of course historical off-line data in the form of traffic traces that may be used to model arrivals of both classes of traffic. In our scheme, in each round of τ seconds, the AC determines two numbers: a rate cap r and the maximum number of senders n . It allocates r to the first n senders seeking permission to send and 0 to any subsequent requests in that cycle.

While variance of the combined traffic from all throttled sources motivated us to use a common rate cap as a control mechanism, the efficacy of the admission control must be measured in terms of throughput and link-overloading. Ideally, link utilization stays very close to the target utilization set by network operators. Thus we would like to penalize deviations from the target utilization. However, exceeding the target capacity can lead to network instability and thus must be penalized more than link

Algorithm 1 AIMD **Input:** average target utilization μ , additive increment a , multiplicative decrement d , current utilization u , current rate cap c , current number of permits n , number of entry requests r **Output:** (rate cap, number of permits)

if $(u > \mu)$ return $(cd, \lfloor nd \rfloor)$
 if $(\delta u > 0)$ return $(\frac{cn+a}{r}, r)$
 return $(\frac{cn}{r}, r)$

underutilization². Moreover, we penalize link underutilization only when there is class-2 traffic that is waiting to be transported. For time running from 0 to T , and with a target capacity c , our loss function is as below:

$$L := \frac{1}{T} \sum_{t=0}^T l(c_1(t), c_2(t), c, b(t))$$

where $b(t)$ is the total buffered traffic at class-2 senders at time t , $c_i(t)$ is the class- i traffic on the link at t , and $l(\cdot)$ is the instantaneous loss function below:

$$l(a, b, c, d) := \begin{cases} \alpha(a + b - c) & \text{if } a + b \geq c \\ \min\{c - (a + b), d\} & \text{else} \end{cases}$$

In the above, α is the penalty factor for exceeding the target capacity. In the simulation experiments below, we used $\alpha = 10000$.

5 AIMD Heuristic

Analogous to the ‘‘additive increase multiplicative decrease’’ heuristic in TCP implementations, we implemented a simple reactive control policy that adjusts the total bandwidth allocated to all requesting class-2 sources up or down based on link utilization feedback. The idea is to increase the total bandwidth additively if the average link utilization u is below a target μ and decrease it multiplicatively otherwise. The total allocated bandwidth is divided equally across permitted senders. The policy has three parameters: target link utilization μ , additive increment a , decrement factor d . The algorithm tracks the current rate cap c , the current number of permits n , and entry requests r received in the prior cycle and calculates the new rate cap and number of permits as displayed in Algorithm 1. Note that AIMD does not increase class-2 allocation unless utilization is below target and utilization gradient is positive. Without the latter condition, rate caps would keep increasing even when currently active class-2 sources do not use up the allocation.

6 Reinforcement Learning

In admission control for Internet traffic management, not only does an AC action (such as additive increment and multiplicative decrement in AIMD in the previous section) result in an immediate cost but it also affects the future of the whole system. For example, setting the rate cap too low may require a long period for the system

² This is reasonable since opt-in controlled sources do not care too much about delays in the discretionary traffic that they send via AC.

to take a sequence of additive increment actions to raise it back to an appropriate level; on the other hand, setting the rate cap too high may be helpful for transmitting all pending class-2 traffic, but can be risky for the next few cycles if class-1 traffic increases substantially. Therefore, optimal control in this problem has a *sequential* nature that requires the controller to take actions that balances immediate and future loss. This problem is naturally modeled as a sequential decision making problem and reinforcement learning (RL) [4] can be applied to optimize the controller.

In sequential decision making, a controller repeatedly chooses an action based on information it observes about the system (called a “state”). After taking the action, the controller receives an immediate loss and observes the next state of the system. We call the tuple (s, a, l, s') a *transition*, where $s \in S$ is a state, $a \in A$ is an action, $l \in \mathfrak{R}$ is the immediate loss, $s' \in S$ is the next state, and S and A are the sets of states and actions, respectively. The goal of RL is to learn from a set of transitions to find an optimal policy, $\pi^* : S \rightarrow A$, to minimize the total loss when π^* is followed. With a discount factor $\gamma \in (0, 1)$, we define

$$Q^*(s, a) = \mathbb{E}[l_1 + \gamma l_2 + \gamma^2 l_3 + \dots]$$

as the expected discounted total loss by taking action a in state s and then following π^* thereafter. The optimal policy π^* can be easily computed if Q^* is known: $\pi^*(s) = \arg \min_{a \in A} Q^*(s, a)$. The Q-learning algorithm is a classic RL method for learning Q^* from observed transitions. Starting from an arbitrary Q-function estimate, Q , the Q-learning algorithm repeatedly refines the estimate when a new transition (s, a, l, s') is observed:

$$Q(s, a) \leftarrow Q(s, a) + \eta \left(l + \gamma \min_{a' \in A} Q(s', a') - Q(s, a) \right)$$

where $\alpha \in (0, 1)$ is a step-size parameter. Under certain assumptions, the estimate Q in Q-learning converges to Q^* .

Q-learning was originally proposed for solving finite-state, finite-action problems. In our network control problem, however, the numbers of states or actions are too large for the algorithm to be tractable, motivating the use of approximation architectures. Here, we require as input a set of k feature functions $(\phi_1, \phi_2, \dots, \phi_k)$, and represent the Q-function by a linear combination of them: $Q(s, a) \approx \sum_{i=1}^k w_i \phi_i(s, a)$. The update formula now becomes

$$\forall i : w_i \leftarrow w_i + \alpha \left(l + \gamma \min_{a' \in A} Q(s', a') - Q(s, a) \right) \phi_i(s, a).$$

In the AC problem, we hand-crafted a few features, such as the average link utilization u , previous rate cap r , and average class-1 traffic volume in a two-month period³.

7 Experiments

We use actual traces of traffic (netflow records) on a link on Yahoo!’s backbone to run simulations and evaluate some admission control policies. We split the records by date for tuning and testing: June 20-25 and Jul 1-3 for tuning and July 11-16 for blind testing. In addition, for reinforcement learning policy we used minute-of-day traffic

³ Due to space constraint we cannot go into much detail, but this feature was helpful for the reinforcement learning policy.

statistics collected over 45 days prior to June 20. We classify sources into class-1 or class-2 based on the source address and source port pair (srcaddr, srcport). Since these traces are obtained on the current network which does not have the proposed admission control mechanism implemented, we must treat class-1 arrivals differently from class-2 arrivals. class-1 arrivals are treated without any modification – as they are seen on the link. However, a series of class-2 flows from the same (srcaddr, srcport) pair that are contiguous in time is pulled together to form a single class-2 request that first arrives at AC at the beginning of the series. For example, from the same (srcaddr, srcport) pair, records such as (t=0, q=1500), (1, 1500), (2,1500), (3,1500) are treated as a single class-2 arrival at $t = 0$ with 6000 bytes to transmit. Since class-1 traffic enters the link directly, all class-1 sources are treated as a single collective class-1 source. Thus the arrivals input to the simulator is a file containing lines such as below, where the first column is the time stamp (1-second granularity), second column is the ID of the sender, and third column is the number of bytes.

```

1243839494 1 17153
1243839494 2.0 185430
... ..
1243839494 2.10 1365260
1243839495 1 25533
1243839495 2.11 68176

```

We tuned AIMD parameters on July 1 data. The traffic pattern for July 1 is shown in Figure 1: with $\tau = 5$, $a = 800$, $d = 0.7$, AIMD achieves a capacity reduction of about 19%.⁴ The AC negotiated class-2 traffic and rate-caps can be seen in Figures 3-4.

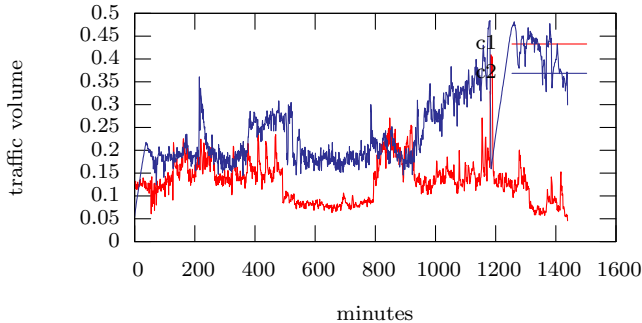


Fig. 3 AIMD-controlled traffic

7.1 Effect of feedback delay

We can expect that a policy that sets rate based on link utilization feedback to perform worse as the feedback delay increases. This is seen in Figure 5. In this figure, each point

⁴ Note that AC-negotiated traffic can see delays longer than 10 seconds and therefore capacity reduction can be bigger than in Figure 2.

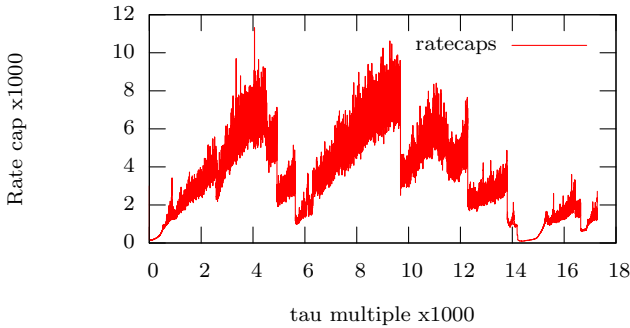


Fig. 4 AIMD Rate Caps

is obtained by loss-tuning parameters for that T for the first day in development test and using the parameters on the second day of the data.

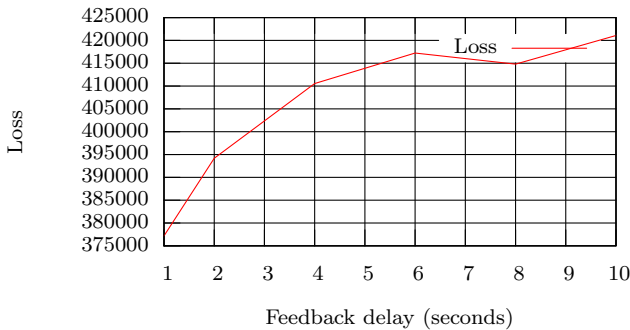


Fig. 5 Effect of feedback delay

7.2 Comparison between AIMD and RL

We tuned RL parameters based on June 20-25 and July 1-3 traffic. We chose 900000 as the target capacity and 700000 as the target average link utilization.⁵ On the development data, we optimized RL and AIMD for different feedback delay values, and measured the average loss of the resulting policies on 12 days of traffic. As shown in Figure 6, reinforcement learned policies have smaller loss than AIMD when feedback delay is small, but larger loss when feedback delay is large. The reduction in loss achieved by RL when feedback delay is no more than 5 is largely gained on days with low volume traffic; their losses are about the same when the traffic is higher volume.

Based on the results in the developmental data, we picked the best policies of RL and AIMD for $\tau = 5$. We then ran each policy exactly once on the test data sets (July

⁵ This pair of parameters is sufficient for AC to transmit almost all class-2 traffic without exceeding the target capacity.

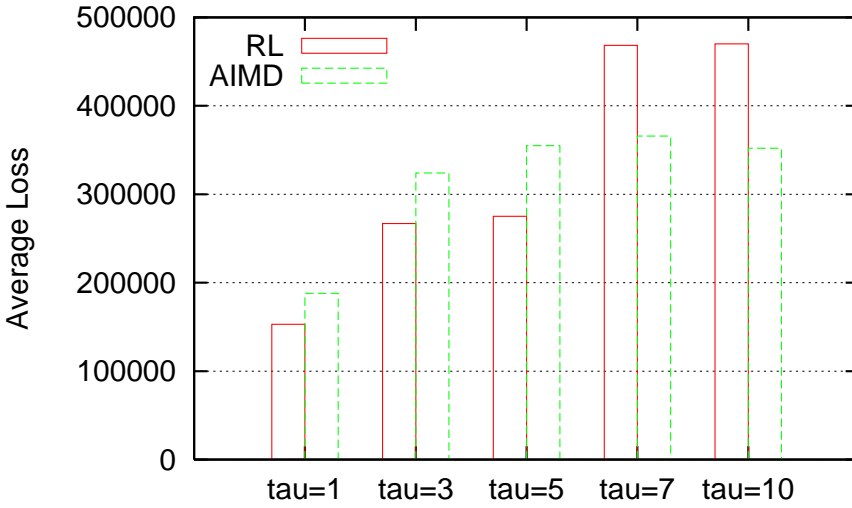


Fig. 6 Average loss of AIMD and RL controller with different feedback delay.

	loss	rate cap	peak util	unsent(%)
AIMD	567194	7468	1107702	0.0005
RL	576761	5824	843742	0.0010

Table 1 Blind test results

11–13 and July 14–16). Table 7.2 reports the average loss, average rate cap, and peak utilization. Both policies gave similar loss, which are consistent with the developmental data’s results (as the test days’ traffics are heavy). The AIMD policy seemed more aggressive, managing to send out more class-2 traffic at the cost of additional loss. On the other hand, the RL policy is more conservative; its average rate cap is significantly lower while avoiding exceeding the target capacity. Due to lack of space, we show just one plot – that of RL-negotiated traffic – in Figure 7.

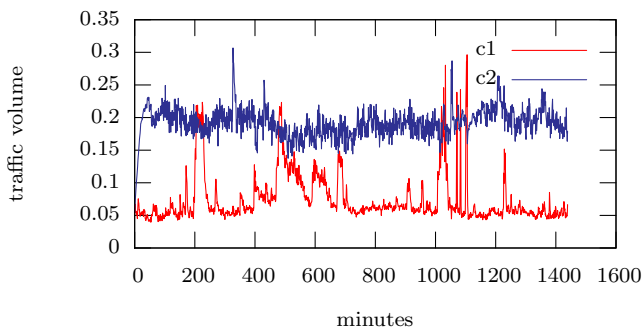


Fig. 7 Reinforcement Learning Rate Caps

Acknowledgements We thank Raymie Stata for introducing us to this problem. We thank Adam Bechtel and Igor Gashinsky for constantly educating us about practical constraints in operating a large network.

References

1. Key, P., Massoulie, L., Wang, B.: Emulating lower priority transport at the application layer: a background transfer service. In: Sigmetrics, pp. 118–129 (2004)
2. Kokku, R., A. Bohra, A., Ganguly, S., Venkataramani, A.: A multipath background network architecture. In: Proceedings of IEEE INFOCOM (2007)
3. Kuzmanovic, A., Knightly, E.W.: TCP-LP: A distributed algorithm for low priority data transfer. In: Proceedings of IEEE INFOCOM (2003)
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
5. Tsugawa, T., Hasagawa, G., Murata, M.: Background tcp data transfer with inline network measurements. In: APCC (2005)
6. Venkataramani, A., Kokku, R., Dahlin, M.: TCP Nice: A mechanism for background transfers. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation. Boston, MA (2002)