Best Effort Session-Level Congestion Control

Siddharth Ramesh
Microsoft Corporation
Redmond, USA
Email: sidram@microsoft.com

Sneha Kumar Kasera
School of Computing
University of Utah
Email: kasera@cs.utah.edu

***Abstract*— Congestion caused by a large number of interacting TCP flows at a bottleneck network link is different from that caused by a lesser number of flows sending large amounts of data - the former would require cutting down the number of competing flows, while cutting down the data sending rate is sufficient for the latter. However, since existing congestion control schemes view congestion only from a packet-level perspective, they treat both to be the same, resulting in suboptimal performance.**

We propose two best effort, search-based, session (or flow) level congestion control strategies for the Internet, to complement existing packet-level congestion control schemes. Our strategies control the number of competing flows to optimize for the flow completion rate and the flow completion time. Furthermore, our session control mechanisms do not require any per-flow state or computation at the routers, make no assumption about input traffic characteristics and requirements, avoid starvation of new flows when existing flows do not leave the system, and do not require any end host TCP modifications. Using evaluations under a wide variety of static and varying traffic load conditions, we demonstrate the significant performance and fairness gains that our session control mechanisms provide.

I. Introduction

In the current Internet, congestion is controlled only at the packet level. There is no mechanism to prevent the build up of a large number of flows at a router. The existing congestion control schemes use packet drops at the routers in conjunction with source back-off to control the number of packets sent towards a congested router link. They neither prevent a large number of flows from contending with each other, nor block new flows from joining the network and further increasing congestion. As a result of this imbalance in the granularity of congestion control, when a large number of flows gain admission into the network, each flow obtains very little bandwidth and stays in the network for a long time. This hurts the flow completion times of elastic applications and the interactivity of streaming applications. Furthermore, it has been shown in [17], [21], [22] that as the number of simultaneous flows increases to a high value, TCP's Additive Increase Multiplicative Decrease (AIMD) algorithm is no longer fair in sharing the bandwidth at a congested link. This is due to the increase in the packet drop rate which further reduces the throughput of flows from their fair share. There is a significant degradation in network goodput as well, because of the increased retransmissions.

This paper addresses the problem of controlling congestion at the session (or flow[1]) granularity in a best-effort manner. The solution we propose is to complement, rather than substitute, packet-level congestion control mechanisms currently in place. As in the case of packet level congestion control, an effective session level control must drop new sessions at the routers in conjunction with session back-off at the sources. However, in this paper, we focus only on session level control at the routers, while relying on TCP's packet level exponential backoff mechanism[2] at the sources. We do not propose any new session control at TCP sources.

The overall functional goals of our session-level control at the router are to maximize the flow completion rate and minimize flow completion time of elastic TCP flows during session overload at routers. Unlike the vast amount of research on flow admission control, our session control is *best-effort* and does not offer any performance guarantees. In addition to these functional goals, our session level control also meets the following logistical and engineering requirements to be practical and deployable.

- Routers do not maintain any per-flow state.
- No *a priori* traffic (or flow) distribution is assumed and no characterization of incoming flows (such as using token bucket parameters, peak rate etc.) is required.
- Our scheme is robust to changes in traffic patterns, both sudden and gradual.
- In the worst case, the performance does not degrade below that obtained without any session control.

Traditionally, overload control is achieved by comparing one or more measures of system or network load against one or more specified load thresholds. The limitation of such threshold-based approaches is that, the specified thresholds, unless tuned, are not likely to result in optimal performance under all network and system conditions. In fact, the performance could be well below optimal when network and system conditions change significantly.

This paper proposes two novel search-based session control strategies that do away with load thresholds. The bottleneck router controls the rate at which new sessions are admitted

[1]We use the terms *session* and *flow* interchangeably throughout this paper. However, we use the term session control to not confuse with flow control.

[2]TCP's packet level exponential backoff mechanism also applies to the first message of a TCP connection. Hence, when a TCP SYN is dropped, an exponential backoff effectively provides session control at the source.

into the network during session overload. The router *searches* through a range space of admit rates and selects the rate that attempts to maximize the overall flow completion rate while minimizing per-flow completion time.

Our first strategy uses a unique combination of Golden Section Search (GSS) and Gradient Ascent (GA) to perform this search. However, we find that for Pareto distributed input flow sizes, maximizing the flow completion rate, using GSS + GA, can result in very high flow completion times. This is because, for Pareto distributed input flow sizes, the maximal flow completion rate is achieved at an admit rate that is higher than the maximum flow completion rate itself. This behavior causes higher flow completion times. Hence, to keep the flow completion time under check, we develop a second strategy that searches for an admit rate that maximizes the flow completion rate but does not exceed the completion rate.

The flow completion rate and hence the optimal admit rate are not fixed. These could change with variations in the flow arrival rate, flow size and available bandwidth. Both of our strategies detect such changes and converge to the new optimal admit rate. This makes our search-based schemes extremely adaptable to changing traffic patterns and network conditions.

We evaluate our session control schemes using extensive ns2 [24] simulations under a variety of stationary and changing traffic loads and find that our schemes result in significant improvements in flow completion rate and flow completion time.

The rest of this paper is organized as follows. Section II reviews other recent work on admission control, specially for elastic flows, and discusses their drawbacks and limitations. In Section III, we motivate the need for session control and discuss some of the important aspects of session control, that lead to the session control algorithms detailed in Section IV. We follow it up with a detailed evaluation of our algorithms under a variety of static and varying load scenarios in Section V. Section VI discusses some of the issues involved with deploying such a scheme at the router. We conclude in Section VII with directions for future work.

II. Related Work

Call admission control has been developed and used extensively in telephone [12] and ATM networks [1], [8], [19], [20], [23]. It has also developed as flow admission control in the IntServ [3], [4], [9], [11] networks. The goal is to reserve (and police) resources as specified by the incoming flows and schedule access to resources to provide statistical or hard performance guarantees. However, because of the variations in demands of Internet applications and the complexity in specifying and policing the per-flow needs, these schemes have not been widely deployed in the Internet. Our session control is best-effort.

Arguments for admission control for elastic traffic were first made by Massoulif and Roberts [14]. Later, Roberts et al. [2], [10] presented a measurement-based admission control mechanism for elastic traffic that estimates the "available bandwidth" in a bottleneck link and admits a new flow only if this estimate is greater than a minimum threshold which any new flow should achieve; else, all new flows are rejected till more bandwidth becomes available. They experimented with two different techniques for estimating the "available bandwidth" - one using a "phantom" TCP connection over the bottleneck link and measuring the bandwidth it receives; the second, measuring the loss probability over the bottleneck link and relating it to the TCP throughput.

Based on the work of estimating QoS parameters of traffic streams in ATM networks using theory of large deviations [5], [7], Mortier et al. proposed a measurement based admission control scheme in [18] for elastic traffic in the Internet. This is a threshold based scheme - the effective bandwidth of the traffic mix is estimated using the entropy of the input traffic and a new flow is admitted only if the packet drop rate does not exceed a certain threshold. Kumar et al. proposed a TCP admission control scheme [13] that uses a link occupancy threshold as the admission control criterion. In this scheme, new flows are admitted only if the link occupancy is below a set threshold.

All the above schemes make assumptions about the nature of incoming traffic in making their admission decision, or use loss thresholds. Although the assumption of knowing the nature of incoming traffic might hold for streaming flows, it does not necessarily hold for elastic flows that send data at variable rates and in an unpredictable manner. Moreover, all of the above schemes are threshold based and hence suffer from the drawback of not being robust against changing network and system conditions.

Our work differs from the previous works in following significant ways. First, instead of using load thresholds, our scheme uses a search-based approach to find the optimal admit rate making the scheme more robust to changing traffic patterns. Second, it does not make any assumptions about the bandwidth requirements of network flows. Third, in addition to an increase in per-flow performance, we show significant improvement in the fairness between short and long lived flows. Last, our scheme does not starve new flows even if existing flows stay in the system for a long time.

III. Motivating Session Control

A. Effect of large number of simultaneous TCP flows

When the number of simultaneous flows increases to a large value, the per-flow throughput is no longer proportionally less - it is worse. This, as we briefly mentioned in Section I, is because TCP's AIMD algorithm no longer emulates Processor Sharing (PS) when the number of competing flows grows to a large value [17], [22]. When the number of TCP flows sharing a link is large (greater than the number of packets in the delay-bandwidth product [17]), the ability of TCP to share the bottleneck link efficiently and fairly decreases. Qiu et al. [22] state that when this happens, only as many connections as the number of packets the network can hold, are active, or achieve goodput considerably larger than zero; the rest of the connections are practically shut-off due to constant timeouts. This implies that as the flow arrival rate increases, the average service rate (and hence, the average goodput) of the flows decreases to even *less* than the fair share.

B. Session control through rate-limiting

We perform a series of simulations using the ns2 simulator to test the above hypothesis and to motivate the need for session control. The simulation network topology consists of a single bottleneck topology (of bandwidth 10 Mbps and delay 100 ms) as shown in Figure 1. There are 10 sources generating new file transfer TCP connections according to a Poisson process. The flow (file) sizes are drawn from a Pareto distribution with a shape parameter of 1.5 and mean of 10000 bytes (typical of web traffic [6]). Though new flows are generated continuously, old flows continue to remain in the system until they complete their transfer. This might not be the case with a human in the loop, with user behavior (and impatience) playing a role in the longevity of a flow. However, for the scope of this work, we discount user behavioral patterns. It is to be noted however, that the arrival rate of new flows in to the system is not a strict function of the network performance, though there could be a loose coupling.

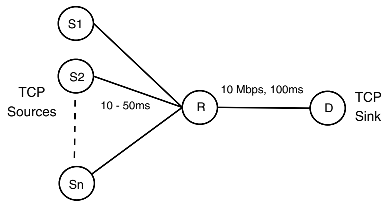

Fig. 1. Simulation topology

The mean arrival rate of new flows at the router is fixed at 1000 flows per second, which corresponds to a high load in terms of the number of sessions. However, the router is engineered to *admit* flows only at a fixed rate in each experiment. Flows are admitted (or discarded) by allowing (or dropping) TCP SYN packets at the bottleneck router. Flows that are dropped retry using the usual TCP retransmit mechanism. We now discuss the effect admission control has on different system parameters.

1) Rate of flow completion: Figure 2 shows the rate of completion of flows as a function of the rate at which flows are *admitted*. For a given network setting and average flow size, the flow completion rate ($\equiv$ number of flows completing in a given interval of time) is maximized for a particular admit rate ($\approx$ 250 flows/s). For any other admit rate, the number of flows completing is suboptimal. We note that when no session control is deployed and all the flows are admitted as they arrive

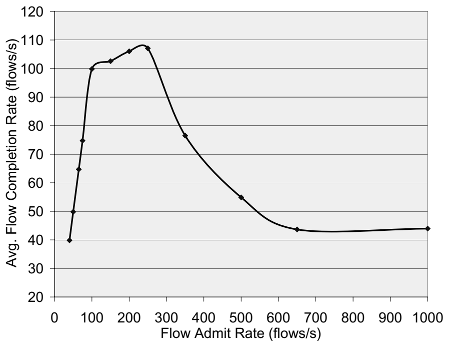

Fig. 2. Effect of flow admit rate on flow completion rate

(this corresponds to the value 1000 on the x-axis), the flow completion rate is less than half the maximum.

2) Per-flow performance: Table I shows the per-flow performance (in terms of average flow completion times and goodputs) of the flows that complete, for different admit rates at the router. As we decrease the rate of flow admission, both the average flow completion times and goodputs improve. This is obvious, since a lower rate of admission corresponds to lesser number of simultaneous flows sharing the bottleneck, and hence, a greater per-flow share.

TABLE I
PER-FLOW PERFORMANCE AT DIFFERENT ADMIT RATES

Flow Admit Rate (flows/s)	Flow Completion Rate (flows/s)	Avg Flow Completion Time (s)	Avg Goodput (Kbps)
1000	43.944	69.178	2.61
250	107.088	42.031	4.46
150	102.6	25.001	8.73
100	**99.84**	**4.053**	**49.83**
65	64.716	**3.0178**	**63.48**

3) Fairness between small and large flows: The flow sizes of the generated flows, as we said earlier, are drawn from a Pareto distribution with a mean of 10000 Bytes and shape parameter of 1.5. For these parameters, the mean corresponds to 81 percentile of the flow sizes; i.e., 81% of the flows are less than 10000 bytes in size. Table II lists the average size of the flows that complete for different rates of flow admission, as well as the percentile corresponding to 10000 bytes. With no session control (corresponding to an admit rate of 1000 flows/s), the distribution of the flows that complete is highly skewed compared to the input traffic distribution. Almost all ($\approx$ 99%) flows that complete are small flows (less than 10000 B). However, with session control through rate-limiting, the output distribution follows the input flow size distribution much more closely - for an admit rate of 100 flows/s, 10000 B corresponds to almost 81 percentile, as should be.

Thus, even if we are not able to control the rate at which new flows arrive at a bottleneck router, just by deploying session control and admitting flows at an *optimal* rate, we can significantly improve the flow completion rate, per-flow performance and the fairness between *mice and elephant* flows.

C. The effect of Pareto

We make an interesting observation from Figure 2. We note that the maximum flow completion rate ($\approx$ 110 flows/s) is achieved at a much higher admit rate ($\approx$ 250 flows/s). This appears counter-intuitive since, even if the system reaches steady-state, the number of flows in the system is building continuously, putting the system under more and more strain. However, this phenomenon can be attributed to the Pareto nature of flow sizes. As mentioned in the experimental setup, the sizes of flows are Pareto distributed (with a mean of 10000 and shape parameter of 1.5). Examining the mean of our Pareto distribution, we find that it corresponds to 81 percentile of flow sizes. This implies that a large number of flows have a flow size below the mean and complete much faster (within a few round trip times). As a result, we can admit at a rate higher than the completion rate and still maintain a high overall flow completion rate.

D. Goals of our session control algorithm

We see that for Pareto flow size distributions, the flow completion rate is maximized at an admit rate that is higher than the completion rate. On the other hand, the per-flow performance (flow completion times and goodputs) is maximized for any admit rate that is less than or equal to the completion rate. These observations lead to two goals : (i) maximizing the flow completion rate, and (ii) minimizing the flow completion time. In this work, we develop two algorithms to meet these goals.

IV. Session Control Algorithms

A. Algorithm 1 - Maximizing the flow completion rate

The goal of this algorithm is to find the admit rate that maximizes the flow completion rate. Further, since the maximum flow completion rate can shift with time (corresponding to changes in traffic patterns and background traffic), the algorithm needs to detect such changes and converge to the new maximum in a reasonable amount of time. The problem of finding the maximum (minimum) is typical of any global-optimization algorithm that maximizes (minimizes) a function - the "function" in this case being the rate of flow completion. In our context, several practical issues make the optimization problem more challenging. First, the "function" in our problem changes with time even if evaluated at the same point, i.e., $f_t(x)$ need not the same as $f_{t-1}(x)$. Further, the number of flows completing during a time interval is not just a function of the number of flows admitted during that interval, but depends on all previous flow admit rates, i.e., if the admit rate during some time interval i is a_{t_i}, the flow completion rate d_{t_i} is not just a function of a_{t_i}, but a function of $(a_{t_i}, a_{t_{i-1}}, a_{t_{i-2}}, ...)$.

In order to search for the optimal admit rate that maximizes the flow completion rate under changing conditions, we develop a new heuristic. Our heuristic uses a combination of two search algorithms - Golden Section Search and Gradient Ascent. Golden Section Search (GSS) is a powerful optimization algorithm that is especially attractive for our context because it does not require the calculations of derivatives of the function it is optimizing.

GSS is quick to reduce the search space considerably by successively narrowing the bounds containing the optimum value by evaluating points on either side of the optimum. However we find that as the range of bracketing gets smaller, GSS causes a lot of fluctuations. To remove these fluctuations, our scheme shifts to the Gradient Ascent algorithm when the bracketing range is smaller than a threshold. Gradient Ascent (GA) is a simple hill climbing optimization algorithm that approaches the local maxima by taking steps proportional to the gradient of the function at the current point, until a peak is found. Since the gradient becomes smaller as the peak is approached, the step size also becomes smaller. Thus GA moves more steadily towards the optimal value and is much less affected by noise and oddities in the measurement of the flow completion rate. Our scheme is detailed in Algorithm 1.

In brief, our algorithm performs the following actions.

1) **Brackets the maximum:**
When the algorithm starts, it must first bracket the peak before GSS takes over. Bracketing of the peak is done in a straightforward manner - the admit rate is successively increased until there is a drop in the flow completion rate, at which point a bracket has been found.
2) **Narrows the bracketing range:**
Golden Section Search successively divides the bracketing interval according to the golden ratio, i.e., the next point chosen is a fraction 0.3819 into the larger interval. GSS continues to reduce the bracketing range until it becomes lesser than a specified threshold (`GSS_THRESHOLD` in Algorithm 1) at which point, it starts to use GA.
3) **Moves steadily towards the peak:**
Gradient Ascent tends towards the peak more slowly than GSS, taking steps whose length is proportional to the gradient of the function at the current point.

TABLE II
FAIRNESS BETWEEN LARGE AND SMALL FLOWS WITH SESSION CONTROL

Flow admit rate (flows/s)	Avg. size of completed flows (Bytes)	Percentile corresponding to 10000 B
1000	4657.86	98.63
200	7491.14	84.27
150	8478.90	82.05
100	10142.48	**80.98**
65	10171.74	**81.14**

Algorithm 1 Maximize flow completion rate

```
$SHIFT(a,b,c,d) : (a) \leftarrow (b); (b) \leftarrow (c); (c) \leftarrow (d)$
$GOLD \leftarrow 1.618$
$C \leftarrow 0.3819$
$\alpha_{min} \leftarrow$ Minimum flow admit rate threshold
$\alpha_t \leftarrow$ No of flows allowed during time interval $t$
$d_x \leftarrow$ No. of flows departed when using admit rate of $x$
Ensure: $\alpha_t \geq \alpha_{min}$
  //Golden Space Search to reduce the search range
  $\alpha_1 \leftarrow \alpha_{min}$
  $\alpha_2 \leftarrow \alpha_1 + stepSize$
  while $d_{\alpha_{t-2}} \leq d_{\alpha_{t-1}} \leq d_{\alpha_t}$ do
    $\alpha_{t+1} \leftarrow \alpha_t + GOLD * (\alpha_t - \alpha_{t-1})$
  end while
  $a \leftarrow \alpha_{t-2}; b \leftarrow \alpha_{t-1}; c \leftarrow \alpha_t$
  while $c - a \geq GSS\_THRESHOLD$ do
    $x \leftarrow b + C * (c - b)$
    if $d_x > d_b$ then
      $SHIFT(a, b, x, b + C * (c - b))$
    else
      $SHIFT(c, x, b, x - C * (x - a))$
    end if
  end while
  //Gradient Ascent
  while TRUE do
    $\alpha_{t+1} \leftarrow \alpha_t + \frac{d_{\alpha_t} - d_{\alpha_{t-1}}}{\alpha_t - \alpha_{t-1}} * stepSize$
  end while
```

4) **Detects changes in traffic pattern:**
The algorithm remains in GA unless it detects a sudden change in the traffic pattern. GA shifts automatically with small and smooth changes in the traffic pattern, thus keeping the algorithm very responsive. However, when the traffic change is sudden and significant, we find that GA does not converge quickly enough to the new peak. Hence, we enhance our heuristic such that, when the algorithm detects such sudden changes, it loops back to GSS and incorporates the faster procedure to localize the region around the new peak.

The α_{min} threshold defines the minimum accessibility of the network link to new flows. It is introduced to prevent the starvation of new flows in gaining admission, when existing flows are long lived and do not leave the network. We defer more discussion on this issue to Section V. The convergence time of the above algorithm is linear in the range space since successive points are chosen linearly with additional function evaluations.

B. Algorithm 2 - Admit rate not exceeding flow completion rate

The primary goal of this algorithm is to keep the flow completion time under check. This is achieved by searching for an admit rate that attempts to maximize the flow completion rate under the constraint that the admit rate does not exceed the flow completion rate. Thus when the admit rate is greater than the completion rate, the algorithm must decrease it to bring it close to the completion rate. We refer to this stage of the algorithm as the *'closing'* phase of the algorithm.

The flow admit rate always acts as an upper bound for the flow completion rate. As a result, even at very low admit rates, the completion rate will be equal to the admit rate. However, at such rates the system might perform sub-optimally because the bottleneck bandwidth might not be used to its full capacity. Hence our algorithm attempts to find the *maximum* flow admit rate at which the admit and completion rates are equal. When there is an increase in the available bandwidth of the bottleneck (due to a decrease in the background traffic), the system can admit and sustain a larger number of simultaneous flows. In order to detect such increases in the available bandwidth, the second stage of this algorithm is to *'probe'* for additional bandwidth.

We use a Multiplicative Increase Multiplicative Decrease (MIMD)[3] algorithm in order to 'close' and 'probe'. Algorithm 2 details our session control scheme. The decrease factor is proportional to the difference between the flow admit and completion rates. Furthermore, the algorithm smoothens the measured flow completion rate using a weighted moving average (with a weight β) to prevent oscillations due to sudden changes in traffic.

Algorithm 2 Admit rate not exceeding flow completion rate

```
$\alpha_{min} \leftarrow$ Minimum flow admit rate threshold
$\alpha_t \leftarrow$ No. of flows allowed during time interval $t$
$a_t \leftarrow$ No. of new flows arriving during time interval $t$
$d_x \leftarrow$ No. of flows departed when using admit rate of $x$
Ensure: $\alpha_t \geq \alpha_{min}$
  //Smoothen flow completion rate estimate with history
  $SFCR \leftarrow (1 - \beta) * SFCR + \beta * d_{\alpha_t}$
  $\alpha_1 \leftarrow a_t$
  if $\alpha_t > SFCR * BUFFER\_FACTOR$ then
    //Multiplicative Decrease
    $\alpha_{t+1} \leftarrow \alpha_t + stepSize * \frac{SFCR - \alpha_t}{SFCR}$
  else
    // Multiplicative Increase
    $\alpha_{t+1} \leftarrow SFCR * INCREASE\_FACTOR$
  end if
```

V. Performance Evaluation and Discussion

In this section, we evaluate both our session control algorithms with the help of simulations using the ns2 simulator.

A. Performance metrics

We mainly concentrate on two metrics of performance - flow completion rate (or equivalently, the total number of flows

[3]Unlike, at the packet level, MIMD does not cause any unfairness at the session level.

completing) and per-flow performance (indicated by average flow completion times and goodputs). In some experiments where it is necessary to show the functioning of our algorithm under changing traffic patterns, we also show the time behavior of the algorithm.

B. Experimental Setup

The simulation topology is as explained in Section III, Figure 1. The bottleneck router buffer is set (in terms of packets) as a product of the bandwidth and the average delay of all flows passing through it. The `stepSize` and `GSS_THRESHOLD` used in Algorithm 1, are set to 20 and 100 respectively. The epoch between successive admit rate evaluation for Algorithm 1 is 2 seconds. For Algorithm 2, `stepSize` is set to 40, `BUFFER_FACTOR` to 1.05 and `INCREASE_FACTOR` to 1.1. The epoch between successive admit rate evaluation for this algorithm is 1 second.

In order to evaluate both our session control algorithms, we have extended the `Queue/DropTail` class in ns2 to a `DropTailSessionControlQueue` class which functions as a drop-tail queue enhanced to implement both our session control algorithms. In all forthcoming plots, we refer to Algorithm 1 as $GSS + GA$ and Algorithm 2 as CP (for Close and Probe). We perform several experiments under both stationary and varying traffic patterns.

C. No change in traffic pattern

In this set of experiments, the traffic is kept stationary such that the optimal admit rate does not vary with time. We observe two important properties of our session control algorithm - convergence and optimality.

1) Increase in flow completion rate: Figure 3 compares the number of flows completing during the simulation run for four cases: no session control, optimal session control, session control using $GSS + GA$ and session control using CP, for two different average flow sizes (10000 and 30000 bytes) and two different flow arrival rates (100 and 1000 flows/s). The optimal admit rates and the corresponding flow completion rates are obtained by trying a large range of admit rates using a brute force method.

There are four important observations to be made from Figure 3. First, when there is session congestion ($\equiv$ 1000 flows/s), there is a huge improvement in the number of flows completing with session control as opposed to no session control. Especially, when the average flow size is 30000 bytes, and the flow arrival rate is 1000 flows/s, the number of flows completed is $\approx$17 times more when using either of our session control schemes. Second, when there is no session-level congestion ($\equiv$ 100 flows/s), neither session control algorithm performs any worse than having no session control. Third, our algorithms perform almost as good as optimal session control under all traffic loads.

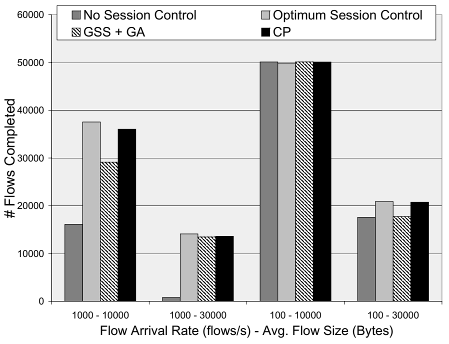

Fig. 3. Comparison of the number of flows completing when the input traffic pattern is not changing with time.

The last and most surprising observation is that even though the primary goal of $GSS + GA$ is to maximize the flow completion rate, it does not perform better than CP. We believe this is because of two reasons. Firstly, the time-behavior of $GSS + GA$ is not very stable. The reason for this is that, while CP always tends towards a known[4] target, the target of $GSS+GA$ (namely, the maximal flow completion rate), is not known and has to be searched. Secondly, as can be observed from Figure 2, the difference between the maximum flow completion rate (target of $GSS+GA$) and the maximum flow completion rate that is equal to the flow admit rate (target of CP) is small. Put together, it is easy for $GSS + GA$ to settle down at a suboptimal flow completion rate.

2) Increase in per-flow performance: Figure 4 shows the smoothened histogram of flow durations with and without session control. The average flow size used is 10000 bytes and the average flow arrival rate is 1000 flows/s. We observe that, with session control, a very high fraction of the completing flows have very small flow completion times as opposed to no session control. Note that this improvement is in addition to the increase in the total number of completing flows due to session control. Further, comparing our two algorithms, it is clear that the improvement is much more for CP than for $GSS+GA$ - a much higher fraction of flows complete with a small flow completion time with CP. This is in tune with the goal of the CP algorithm to improve per-flow performance.

3) Time behavior of flow admit and completion rates: In order to study the working of our algorithms, we now plot (in Figure 5) the variation of the flow admit and flow completion rates for no session control, and session control using $GSS + GA$ and CP.

The top plot shows the time-line behavior when no session control is being employed at the router. Note that even though the flow admit rate is much higher than the completion rate, the completion rate stabilizes at around 50 flows/s. The middle plot corresponds to that obtained when employing $GSS + GA$. The initial part of the $GSS + GA$ algorithm (shown

[4]CP always tends towards the current flow completion rate.

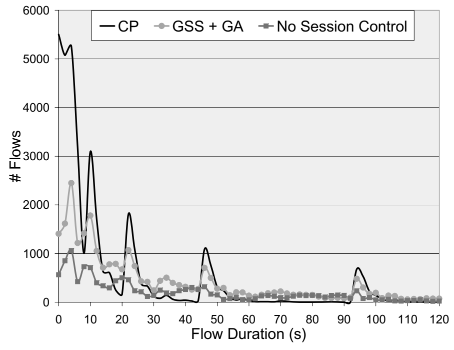

Fig. 4. Histograms of flow durations with and without session control, when the input traffic pattern is not changing with time

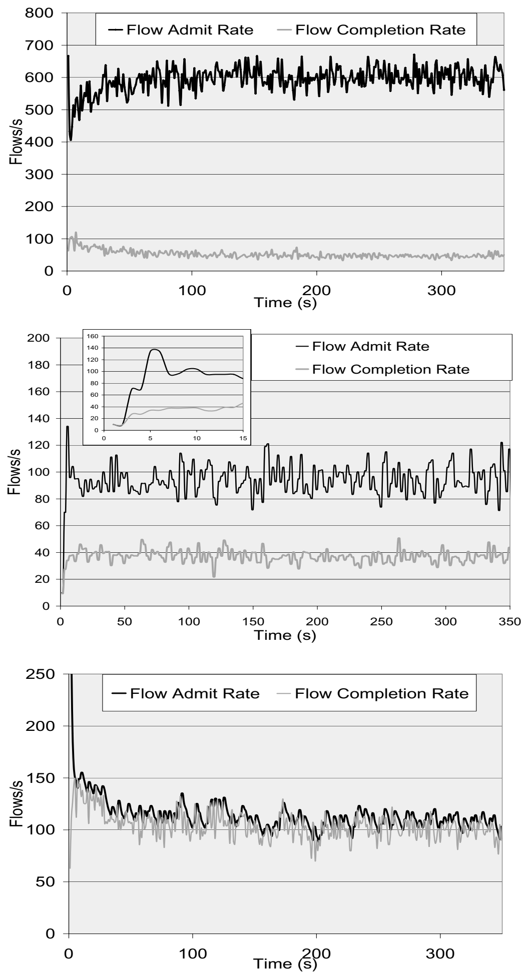

Fig. 5. Variation of flow admit and completion rates with time, when there is no change in the input traffic pattern and (i) *no session control* being employed (top), (ii) GSS+GA is employed (middle) and (iii) CP is employed (bottom).

enlarged in the embedded figure) shows the Golden Section Search part of the algorithm. After sufficient reduction in the search space, Gradient Ascent takes over and the variation in the admit rate is smoother (though in this figure, the variation during the GA phase is higher than normal). Note that even $GSS+GA$ admits at a rate higher than the completion rate, in an attempt to maximize the flow completion rate. The bottom plot shows the behavior of the CP algorithm. The flow admit rate plot is almost indistinguishable from the flow completion rate plot, hinting that the algorithm is doing a good job in finding the maximum admit rate closest to the completion rate. Also note the regular spikes in the admit rate that correspond to the algorithm trying to *probe* for additional bandwidth to admit more flows to check if the completion rate increases proportionally.

D. Effect of changes in traffic pattern

We also experiment with a variety of changing traffic patterns and network conditions to observe how our session control algorithm reacts to such changes. We vary three network and traffic parameters - flow arrival rate, average flow size and bottleneck capacity. For each of these three broad categories, we vary the rate at which the parameters are changing from sudden to gradual. Figure 6 shows the different load change scenarios considered for each traffic parameter. We also experiment with pulse-like variations (Figure 6, bottom right graph) in the three traffic parameters. However, due to lack of space, we are not able to present the results of those experiments in this paper. The details are presented in [25].

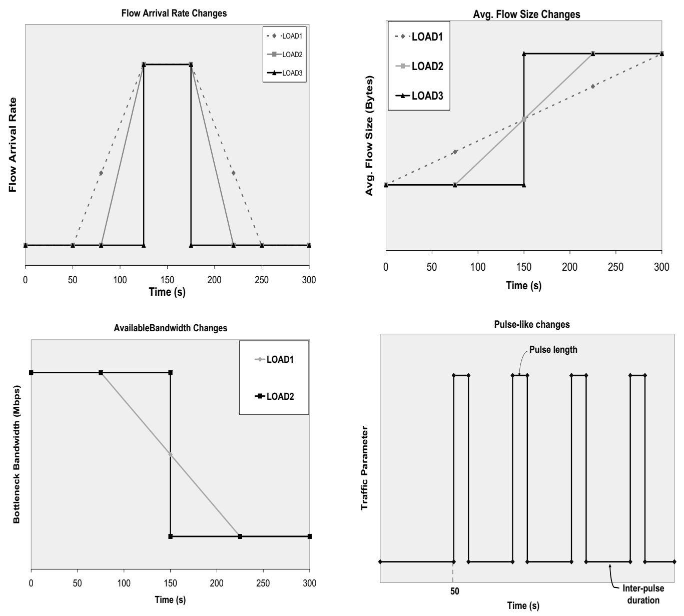

Fig. 6. Different load change scenarios considered.

E. Changes in flow arrival rate

Even though we have argued (and shown) that the input flow arrival rate does not have any effect on the optimality of the flow admit rate, it is interesting to see how our algorithms react to sudden changes in input load caused by changes in the arrival rate of new flows. We perform various experiments with varying degrees of load change. The top left plot of Figure 6 shows the different rates of changes that we consider - from gradual ($LOAD1$) to sudden ($LOAD3$). The experiments are each run for 300 seconds and the flow arrival rate is varied from 100 to 1000 flows/s.

Figure 7 and Table III compare, respectively, the number of flows completing and the average flow completion times, under different load scenarios. The benefit of employing session control is evident. Further, CP performs best, with respect to both criteria.

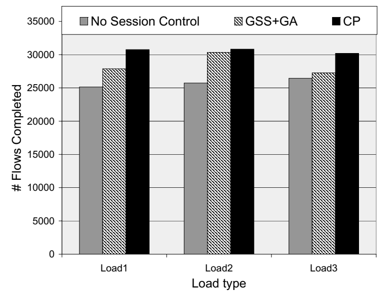

Fig. 7. Comparison of the number of flows completing when the input flow arrival rate changes.

TABLE III
AVERAGE FLOW COMPLETION TIMES WITH CHANGE IN FLOW ARRIVAL RATE

Load type	Avg Flow Completion Time (s)		
	No SC	GSS+GA	CP
Load1	29.56	27.77	**19.92**
Load2	28.32	24.06	**19.39**
Load3	28.64	24.97	**19.66**

1) Time behavior of the algorithm: Figure 8 compares the time behavior of the flow admit and completion rates with and without session control. The load-change condition is that of Load-3 - there is a sudden increase in the flow arrival rate from 100 flows/s to 1000 flows/s at time t = 125 s. The flow arrival rate jumps back to 100 flows/s at t = 175 s.

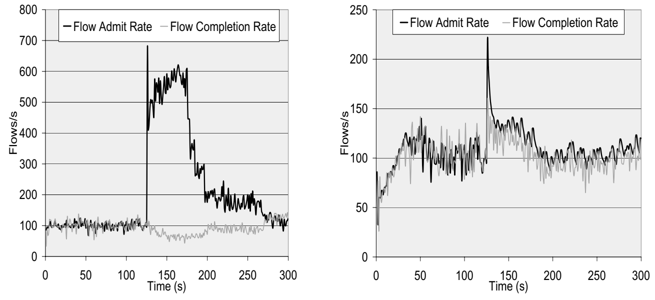

Fig. 8. Time variation of the flow admit and completion rates for changes in the input flow arrival rate with no session control (left) and CP session control (right). The right plot is magnified to observe the effect better.

When no session control is employed, for the first 125 seconds, the flow completion rate is fairly stable and close to the flow arrival rate ($\approx$ 100 flows/s), indicating no session overload. However, from 125 to 175 seconds, the admit rate shoots to very high values due to increase in the flow arrival rate and the lack of any session control. During this time interval, corresponding to this increase in flow admit rate, there is a dip in the flow completion rate. When the flow arrival rate returns to session underload after t = 175 s, the flow completion rate also returns to a value of around 100 flows/s. The dependence of the rate of flow completion on the rate of flow arrival when there is no session control, is obvious from this plot.

On the other hand, when CP session control algorithm is employed the flow admit rate closely follows the flow completion rate throughout the run time of the experiment. Even when there is a spurt in the flow admit rate at t $\approx$ 125 s, it is only because of an increase in the flow completion rate. The increase in the flow admit rate does not affect the flow completion rate in any negative way - the flow completion rate continues to remain greater than 100 flows/s. As a result of the flow admit rate not exceeding what the bottleneck can 'sustain' at any point of time, the flows which do get admitted get good throughput, which is reflected in the low average flow durations seen earlier.

F. Changes in average flow size

In this section, we vary the average flow size of the input traffic, the distribution still being Pareto. Figure 6 shows the different load scenarios considered. The experiment is run for 300 seconds. The average flow size is changed from 10000 bytes to 30000 bytes. We experiment with two flow arrival rates - 100 flows/s and 1000 flows/s.

TABLE IV
NUMBER OF FLOWS COMPLETING WITH CHANGE IN AVERAGE FLOW SIZE

Flow Arrival Rate (flows/s)	Load type	# Flows Completed		
		No SC	GSS+GA	CP
100	Load1	19956	20041	**20625**
	Load2	20682	20574	**21372**
	Load3	20433	21440	**21461**
1000	Load1	6991	**21246**	20408
	Load2	9358	**25209**	21854
	Load3	10944	**25763**	23425

TABLE V
AVERAGE FLOW COMPLETION TIMES WITH CHANGE IN AVERAGE FLOW SIZE

Flow Arrival Rate (flows/s)	Load type	Avg. flow completion time (s)		
		No SC	GSS+GA	CP
100	Load1	16.12	16.51	**12.73**
	Load2	10.18	10.13	**7.27**
	Load3	7.84	9.43	**4.65**
1000	Load1	66.15	39.13	**24.70**
	Load2	64.87	67.41	**23.53**
	Load3	62.50	68.27	**23.31**

Tables IV and V compare the number, and average flow completion times of flows completing with and without session control. A flow arrival rate of 100 flows/s does not load the system when the average flow size is 10000 bytes. However, when the average flow size triples to 30000 bytes, it puts the system under session congestion. Flow arrival rate of 1000 flows/s corresponds to session overload under both flow size distributions. As is expected, there is an overall improvement in performance in employing session control as opposed to no session control.

G. Changes in bottleneck capacity

The optimal admit rate for a given traffic mix would also depend on the available capacity at the bottleneck link. This is because the available bandwidth determines how many simultaneous flows can share the link and still maintain a high performance. Higher the background traffic, lower the available bandwidth, lower the maximum number of simultaneous flows possible and smaller is the optimal admit rate. To observe the effect background traffic has on the optimal admit rate (and the convergence of our algorithms), we experiment with changes (both increase and decrease) in the bottleneck capacity during the simulation run. Once again, we vary the rate at which the changes occur (Figure 6). Since, in reality, background traffic takes up precious router buffer space also, in our experiments we change the buffer capacity along with changing the bottleneck link capacity. We experiment with both increase (10 to 20 Mbps) and decrease (10 to 5 Mbps) in the available bottleneck capacity. However, due to lack of space, we report results only for the more interesting case of decrease in capacity. Figure 9 shows the number of flows completing for a flow arrival rates of 100 and 1000 flows/s.

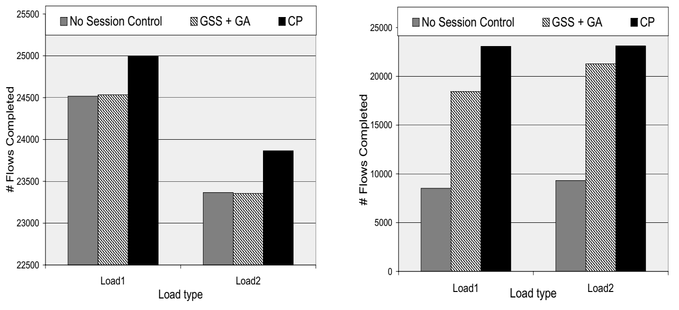

Fig. 9. Comparison of the number of flows completing, with a decrease in the available bottleneck bandwidth from 10 to 5 Mbps. Flow Arrival Rate is 100 flows/s for the left plot and 1000 flows/s for the right plot.

Table VI compares the average flow completion times for different flow arrival rates under different load changes. When the flow arrival rate is low (100 flows/s), the per-flow performance is approximately the same across all algorithms, across all load conditions. One of the reasons for CP not performing as well is because CP takes a longer time to reach its stable flow completion rate until which it admits less flows even in the case of underload. However, CP does well to cope with the decrease in bandwidth and admits only at the new completion rate. For example, for $Load2$, if we consider only the flows that complete in the last 100 seconds (i.e., after the available bandwidth has reduced to 5 Mbps), the average completion times with no session control is 22.6 s, while it is a mere 14.24 s when employing CP.

TABLE VI
AVERAGE FLOW COMPLETION TIMES WITH DECREASE IN BANDWIDTH

Flow Arrival	Load	Avg Flow Completion Time (s))		
Rate (flows/s)	type	No SC	GSS+GA	CP
1000	Load1	48.70	46.30	**19.42**
	Load2	47.79	52.91	**19.5**
100	Load1	**5.64**	5.81	6.15
	Load2	6.98	**5.97**	6.17

VI. Deployment Issues

In this section, we discuss some important issues pertinent to deploying our scheme at the routers. Since congestion is usually present at the access links (and not the links in the backbone of the network that are sufficiently over-provisioned), we envision deployment of our scheme mainly at border routers that connect companies / organizations with their ISP. However, our scheme is amenable to being deployed at the core routers also. As mentioned in Section I, our scheme has a lot of advantageous features making it suitable for deployment.

A. Admission Control

Admission control of TCP flows (i.e., blocking of new sessions) can be done in many ways [18] including dropping SYN packets, or sending RST or ICMP SourceQuench packets back to the source. We use dropping of TCP SYN packets[5] for session control since it incurs smaller amount of overhead in comparison to creating and sending RST or ICMP packets. Routers in our scheme, measure the flow completion rate by tracking FIN packets traversing through them. However, a router may see the SYN of a connection but not the corresponding FIN, or vice versa. This would skew the flow completion rate measure at the router. However since such events are rare in the current Internet [16], we believe that this will not pose a serious problem to the working of our scheme. Moreover, since we are measuring only the rate of connection departure (and not an absolute value of the current number of flows in the system), a skew incurred during one time interval, will not accumulate and increase with time.

Since our algorithms require to snoop into the transport header of each packet to check if it is a connection initiation (SYN) or termination (FIN) packet, they would obviously skip packets whose transport headers are encrypted (such as IPSec packets) or encapsulated (such as IP in IP). Currently, less

[5]In scenarios where data transfer is mainly from a server to a client dropping SYN+ACK might be more appropriate.

than 0.15 % of all packets are IPSec packets [15]; hence this is not a very big hurdle for our scheme. However, it might become more important in the future with increase in IPSec traffic.

B. UDP Flows

In this work, we focus on session control for elastic TCP flows only and not for UDP flows. Since UDP flows are not bound by any packet level congestion control mechanisms, we believe that controlling congestion at the session-level alone would not yield any benefit. Moreover, since UDP flows do not necessarily have identifiable connection setup and teardown messages, keeping track of the arrival and departure of UDP flows will involve maintaining (and updating) per-flow state at the router. This is clearly undesirable. However, we do account for the impact of UDP traffic on the network, when dealing with TCP session control. The presence of UDP flows manifests itself as a reduction in the bottleneck link capacity and an increase in router buffer occupancy. We showed in Section V how our algorithm adapts to changing bottleneck capacity.

VII. Conclusion

In this paper, we have introduced two novel search-based session-level congestion control mechanisms ($GSS+GA$ and CP) to complement existing packet level mechanisms. Detailed evaluations of both our session control algorithms under a variety of traffic scenarios and load changes showed that employing our session control algorithms has huge benefits over no session control during session congestion. Comparing both our algorithms, we find that overall, CP performs better than $GSS+GA$, not only in terms of significantly reducing the flow completion time, but also in terms of maximizing the flow completion rate. Furthermore, it is fairly robust to all changes in traffic patterns including pulse-like variations [25]. Hence, we recommend CP as a viable session control mechanism for the Internet.

An important aspect of session control which we believe will aid in significant performance improvement is “remote overload control”, or session level control at the end hosts that works in conjunction with router session control. In existing mechanisms, the loss of a TCP SYN message results in a backoff of a few seconds before the SYN is retransmitted. This backoff time is increased when consecutive SYNs are lost. Surprisingly, there is no comprehensive study of the timescales of session level congestion or of the choice of these back-off intervals. We will study session level congestion timescales and research changes in the TCP SYN retransmission mechanism to make it more aware of the session level congestion in the network.

References

[1] J. Appleton. Modelling a connection acceptance strategy for ATM networks. *Broadband Technologies*, October 1990.

[2] N. Benameur, S. B. Fredj, S. Oueslati-Boulahia, and J. W. Roberts. Quality of service and flow level admission control in the internet. *Computer Networks*, 40(1):57–71, 2002.

[3] C. Casetti, J. F. Kurose, and D. F. Towsley. A new algorithm for measurement-based admission control in integrated services packet networks. In *PfHSN '96: Proceedings of the TC6 WG6.1/6.4 Fifth International Workshop on Protocols for High-Speed Networks V*, pages 13–28, London, UK, UK, 1997. Chapman & Hall, Ltd.

[4] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 14–26, New York, NY, USA, 1992. ACM Press.

[5] S. Crosby. Statistical properties of a near-optimal measurement-based cac algorithm, 1997.

[6] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.

[7] N. G. Duffield, J. T. Lewis, N. O'Connell, R. Russell, and F. Toomey. Entropy of ATM traffic streams: A tool for estimating QoS parameters. *IEEE Journal of Selected Areas in Communications*, 13(6):981–990, 1995.

[8] Z. Dziong, M. Juda, and L. G. Mason. A framework for bandwidth management in ATM networks - aggregate equivalent bandwidth estimation approach. *IEEE/ACM Trans. Netw.*, 5(1):134–147, 1997.

[9] S. Floyd. Comments on measurement-based admissions control for controlled-load services, 1996.

[10] S. B. Fredj, S. Oueslati-Boulahia, and J. Roberts. Measurement-based admission control for elastic traffic. In *ITC 17*, December 2001.

[11] S. Jamin, P. B. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. *SIGCOMM Comput. Commun. Rev.*, 25(4):2–13, 1995.

[12] S. Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley, Reading, MA, 1997.

[13] A. Kumar, M. Hegde, S. Anand, B. Bindu, D. Thirumurthy, and A. A. Kherani. Nonintrusive TCP Connection Admission Control for Bandwidth Management of an Internet Access Link. *IEEE Communications Magazine*, 38(5):160–167, 2000.

[14] L. Massoulie and J. Roberts. Arguments in favour of admission control for TCP flows.

[15] S. McCreary and K. Claffy. Trends in wide area ip traffic patterns - a view from ames internet exchange. In *ITC Specialist Seminar, 2000*, September 2000.

[16] P. Molinero-Fernandez and N. McKeown. Study of routing behaviour through traceroute measurements. *Network Analysis Times, NLANR*, 1(2), April 2001.

[17] R. Morris. TCP behavior with many flows. In *ICNP '97: Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, page 205, Washington, DC, USA, 1997. IEEE Computer Society.

[18] R. Mortier, I. Pratt, C. Clark, and S. Crosby. Implicit admission control. *IEEE Journal on Selected Areas in Communication*, 18(12), 2000.

[19] N.M.Mitrou and D.E.Pendarakis. Cell-Level Statistical Multiplexing in ATM Networks: Analysis, Dimensioning and Call-Acceptance Control w.r.t. QoS Criteria. *Queueing, Performance and Control in ATM*, pages 7–12, 1991.

[20] P. B. Key. Connection Admission Control in ATM networks. *Broadband Technologies Journal*, 13(3), July 1995.

[21] C. M. D. Pazos, J. C. Sanchez-Agrelo, and M. Gerla. Using back-pressure to improve TCP performance with many flows. In *IEEE INFOCOM (2)*, pages 431–438, 1999.

[22] L. Qiu, Y. Zhang, and S. Keshav. On individual and aggregate TCP performance. In *ICNP '99: Proceedings of the Seventh Annual International Conference on Network Protocols*, page 203, Washington, DC, USA, 1999. IEEE Computer Society.

[23] R.J. Gibbens, F. P. Kelly, and P. B. Key. A decision-theoretic approach to call admission control in ATM networks. *IEEE J. Select. Areas Commun.*, 13:1101–1114, August 1995.

[24] S. McCanne and S. Floyd. ns Network Simulator, 1995.

[25] S. Ramesh. Session Level Congestion Control for the Internet, http://www.cs.utah.edu/∼sramesh/sid-thesis.pdf.