Internet Engineering Task Force Integrated Services WG
INTERNET-DRAFT
S. Jamin/C. Jin/L. Breslau
UMich/UMich/Xerox
draft-ietf-intserv-control-flow-01.txt
Oct, 1997
Expires: 4/15/98

A Measurement Based Admission Control Algorithm for Controlled-Load Service with a Reference Implementation Framework

Status of this Memo

This document is a product of the Integrated Services working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at int-serv@isi.edu and/or the author(s).

Abstract

Controlled-Load Service provides data flows with an enhanced quality of service, in the form of low packet delay and a low probability of packet loss even under congestion. A network element providing Controlled-Load Service can use an admission control algorithm to limit the number of data flows receiving the service. In this document we describe an admission control algorithm for Controlled-Load Service. This algorithm is not intended for IETF

standardization. Rather, it is presented for informational purposes only.

We also present a reference implementation framework for the measurement-based admission control algorithm. Our implementation separates the measurement from the actual admission control decision to provide the flexibility of using different algorithms in bandwidth estimation and admission control.

Introduction

Controlled-Load Service (CL), as defined in [Wro95], is an enhanced quality of service intended to support applications requiring performance better than that provided by traditional best-effort service. Even under congestion, network elements offering CL are expected to provide flows with low delay and low packet loss.

In order to provide this enhanced level of service, network elements must limit the number of flows receiving the service. This can be accomplished by requiring applications to make explicit requests for service. Explicit requests for service can be made using a reservation setup protocol, such as RSVP [B+96], or some other means. Each network element that receives a request for service can either accept or reject the request. We refer to this decision as "admission control."

An application requesting CL presents the network element with a traffic descriptor to describe its data flow. This descriptor includes a token bucket filter and a peak rate. The token bucket parameters, a rate and bucket depth, represent a loose upper bound on the new data flow. A measurement based admission control algorithm (MBAC) admits or rejects a new flow based on measurements of existing traffic and the parameterized description of the new flow. The dependence of MBACs on traffic measurements makes the quality of the service they provide subject to statistical fluctuation of traffic. We expect MBACs to work well only when there is a high degree of statistical multiplexing of uncorrelated flows and traffic fluctuation is not dominated by a small number of flows. In this document, we describe one such MBAC designed for CL.

Admission control is not an area appropriate for IETF standardization. Rather, vendors and service providers are free to implement and deploy any admission control algorithm that enables a network element to meet the service requirements of the Controlled-Load specification. Indeed, admission control can be seen as an area for product differentiation. Hence, the algorithm described here is

presented for informational purposes only, providing a single example of an MBAC that may be used as a reference algorithm.

Various MBACs suitable for use with CL have been proposed in the academic literature. See, for example, algorithms described in [Flo96, JSD97, GK97]. The algorithm described here was first proposed in [JS97] and was shown to perform as well as several other MBACs. This algorithm is designed to be very simple to implement. We believe that it meets the requirements given in the CL specification, performs as well as other known algorithms, and provides sufficient configuration parameters to allow it to be deployed in a variety of settings. We refer the interested readers to the above references both for further details on the other MBACs and for more background on the MBAC described here.

The remainder of this document is organized as follows. In the next section we describe the admission control algorithm. Next, we describe one measurement process that may be used to provide load estimates that are used as inputs to the admission control algorithm. After discussing the different tuning parameters that allow the algorithm to be used in various settings, we present a reference implementation framework of the algorithm.

The Admission Control Algorithm

Our admission control algorithm takes as input L, a load estimate produced by the measurement process (described in the next section), C, the link bandwidth, upsilon, a user defined aggregate loading factor, kappa, a user defined new flow effect factor, and r, the token bucket rate of the new flow requesting admission. Whenever a new flow requests admittance under CL, the flow is admitted if the following inequality is satisfied:

$$L < \text{upsilon} * C - \text{kappa} * r$$

Otherwise the flow is rejected.

The Measurement Process

The purpose of the measurement process is to compute an estimate of the network load attributed to data packets receiving Controlled-Load Service. This estimate, which we refer to as L, is used as input to the admission control algorithm. We describe a time window measurement process here. An alternative measurement process using

exponential averaging may be used instead [Flo96].

The time window measurement process uses 2 parameters, T and S. T is the measurement window and S is the sampling period, with S <= T.

During every sampling period, S, an average load is computed. This average load is simply the sum of bits in packets receiving CL divided by the length of the sampling period. We note that computing average load for a given sampling period is basic to most measurement processes advocated for use with MBAC.

The only per-packet action required by the measurement process is to accumulate the bit-count of packets receiving CL service. All other processing occurs with low frequency. For performance enhancement, a router vendor may wish to implement the per-packet bit counting in hardware. At each operator-defined sampling period S, a software process reads and clears the hardware accumulator. The software process also performs the other low frequency processing to compute the load estimate.

The load estimate, L, is updated as follows:

1. At the end of every measurement window, T, L is set to the highest average load computed for any S during the previous window.

2. If a newly computed average load for a given sampling period S is larger than the current value of L, L is set to the newly computed average.

3. Whenever a new flow is admitted, the measurement estimate is immediately increased by r, the token bucket rate of the newly admitted flow.

The Parameters

In this section we discuss how each of the parameters can be adjusted to control the behavior of the algorithm. The specific settings that are appropriate in any deployment environment depend on the characteristics of that environment (i.e., the traffic characteristics and link bandwidth), on how much Controlled-Load traffic a network operator wants to admit on a link, and on the level of risk the network operator is willing to take that the service requirements are occasionally violated.

T -- Measurement Window

Increasing T increases the amount of history remembered by the

measurement process. The values of T will be some integral multiple of the value of S.

S -- Sampling Period

For a fixed T, decreasing S makes this measurement process more sensitive to bursts of data. Appropriate values of S are likely to be on the order of thousands of packet transmission times.

upsilon -- Aggregate Loading Factor

Upsilon controls the amount of the link resources that can be used by CL traffic. Decreasing upsilon makes the admission control algorithm more conservative and reduces the number of CL flows admitted on a link. Network operator willing to commit all their link capacity to CL traffic might want to start off setting upsilon to 0.7. Depending on the burstiness of extant traffic, upsilon may be tuned to values higher than 1. Larger values of upsilon decreases the "safety margin" of slack bandwidth that may be used to accommodate sudden bursts in traffic. Hence network operators that operate their network with high upsilon run a higher risk of violating CL service description.

kappa -- New Flow Effect Factor

Kappa reflects the network operator's assessment of the effect new flows may have on traffic load. Kappa of 1 provides for the worst case where a new flow may send data at a constant bit rate consummate with its token rate.

Network service providers should have the ability to control the settings of each of these parameters, conditioned upon the network link speed, extant traffic characteristics, and the providers' goals (i.e., the percentage of bandwidth set aside for other services such as best-effort, the degree of risk aversion, etc.). Network operators will need to monitor the performance of the algorithm over time and adjust these parameters to meet changing traffic characteristics and service requirements. Automatic tuning of these parameters is also possible [CKT96].

We mentioned in the Introduction that MBAC works well only on links with high degree of statistical multiplexing where current traffic measurements are reasonable predictors of future load. For links with low degree of statistical multiplexing, the algorithm presented here may be used without the measurement part, for example by maintaining L as the sum of the token rates of all admitted flows, with the parameters upsilon and kappa both set to 1.

Reference Implementation Framework

```
                           +-----------+
                           |           |    +-------------+
                           | Admission |<---| ADC Control |
       +=============+     |  Control  |    +-------------+
       | Reservation |<--->|           | +-------------------+
       +=============+     +-----------+ | Estimator Control |
              ^                          +-------------------+
              |               USER                  ^
   ***********V*************************************V********
              ^               KERNEL                ^
    flow and  |                               +-----V-----+
    class     |            +===========+      | Estimator |
    management|            |           |      +-----------+
        +=====V======+     | Scheduler |            ^
        | Classifier |---->|           |---+        |
        +============+     +===========+   |  +-----V-------+
              ^                  |         +-->| Measurement |
              |                  |            +-------------+
              |                  V
      incoming packets     outgoing packets
```

Figure 1: Overview of the MBAC

We present in this section a description of an implementation of MBAC. This description represents our on-going efforts to implement MBAC on several BSD-derived UNIX platforms. A guiding principle of our implementation is to put components of the architecture that require high-frequency updates in the UNIX kernel, leaving the rest in user space.

Figure 1.0 is an abstraction of the implementation shown with the other integrated services modules, i.e. packet classifier, scheduler, and a reservation daemon, which we expect to be present on the system. Inside the kernel, the classifier intercepts each output packet and determines the output queue to which the packet belongs. The scheduler selects the next packet and dispatch it to the output interface whenever the interface is ready to transmit a new packet. The user level reservation daemon makes new bandwidth reservations or deletes existing ones. The remaining five functional units are part of the MBAC, consisting of a measurement unit and an estimator unit in the kernel, and the ADC (ADmission Control) unit on the user level. The measurement unit counts the total number of bits sent through each interface; the estimator unit computes bandwidth usage estimates for use in admission control equations. These two functions require access to low level network data structures and

need to make frequent computations, which introduces tremendous overhead if implemented on the user level. The user level module includes ADC, ADC Control, and Estimator Control. The ADC unit implements the actual admission control algorithm. The other two units, ADC Control and Estimator Control, allow users to tune the parameters of the admission control algorithm and bandwidth averaging techniques used in the kernel.

The Measurement unit inside the kernel maintains an accounting of the number of bits sent through each interface. It adds to the count of bits sent whenever a packet is dispatched by the scheduler.

```
+---------------------+           +-------------+
| meas_readresetCLM() |<----?---->|             |
+---------------------+           |             |
+---------------------+           +-------------+
| meas_updateCLMq()   |---------->|  CLM_entry  |
+---------------------+           +-------------+
+---------------------+           |             |
| meas_newCLM()       |---------->|             |
+---------------------+           +.............+
```

Figure 2: the Measurement Unit

The measurement unit consists of three interface functions and a data structure, CLMq (Controlled Load Measurement queue.) The details of this unit are shown in Figure 2. The CLMq maintains an entry for each Controlled Load class. In the simplest case there will be one CL class per interface. In the presence of link-sharing, each share can have its own CL class. The structure of each entry is shown as a CLM_entry type in C language:

```
typedef struct  _CLM{
        ClassID         *cid;
        unsigned long   bits_sent;
        unsigned int    multiplier;
} CLM_entry;
```

The first member is a pointer to a ClassID by which the entries in the CLMq is addressed. The ClassID of a CLMq entry associates the entry with its respective CL queue the Scheduler maintains. Since the Scheduler is not part of our architecture, we assume no prior knowledge of the data structures it uses and hence keep only a pointer for a class ID. The member bits_sent is incremented by the packet size in bits whenever a packet belonging to the current class is dispatched by the scheduler; the member multiplier is provided as a safety factor in case the number of bits sent exceeds a 32 bit long integer. There is currently no support for queueing delay

measurement.

The first interface function meas_updateCLMq is invoked by the scheduler after it sends a packet out to a network interface. meas_updateCLMq updates the CLMq entry according to the class of the packet. Function meas_readresetCLMq is provided to the rest of the system as an interface to access the CLMq; it reads, or reads and resets, members of a CLMq entry. The last of the three, meas_newCLM adds an entry for a new class to the end of the CLMq.

```
     +-----------------+ \    +-----------+
 +-->| est_estimator() |  \   |           |
 |   +-----------------+   \  +-----------+
 |   | est_readmeas()  |    > | est_entry |
 |   +-----------------+   /  +-----------+
 |   | est_readest()   |  /   |           |
 |   +-----------------+ /    +-----------+
 |
 |               +-----------------+     Estimator Queue
 +---------------| est_change_fp() |
                 +-----------------+
```

Figure 3: Estimator Unit inside the Kernel

The Estimator inside the kernel is illustrated in Figure 3. It is invoked periodically to compute the sample and average bandwidth usage estimate. A function est_change_fp and an estimation queue constitute the estimator unit. The function est_changefp can change the estimation algorithm for any class; this is necessary since different classes may have different flow characteristics. The estimation queue is organized simply as an array. The structure of an entry in an estimation queue is shown as a structure in C language:

```
        typedef struct _est{
                ClassID *       cid;
                unsigned long   bandwidth_avg;
                unsigned long   bandwidth_var;
                unsigned int *est_estimator(ClassID *, void *);
                unsigned int *est_readmeas(ClassID *, void *);
                unsigned int *est_readest(ClassID *, void *);
        } est_entry;
```

Each entry in the queue stores the average and the variance of the bandwidth usage. The function pointer est_estimator points to the actual estimation routine that calculates quantities such as bandwidth usages or queueing delays. Currently only bandwidth usage estimation is supported, but we allow for extension to estimate other

flow characteristics in the implementation framework. The function est_readmeas allows access to the raw measurement samples and the function est_readest allows access to the estimate. The user level processes can thus access both results of estimation calculation and the raw data in the CLMq through a system call.

```
                        +-----------------+
                        | adc_changealgor |
                        +--------^--------+
                                 |
      +-------------+   +--------V--------+   +-----------------+
      | Reservation |<->| adc_algorithm[] |<->| est_changeparam |
      +------^------+   +-----------------+   +------^----------+
             |                                       |
         +---V---------------------------------------V-----+
         |                     KERNEL                      |
         +-------------------------------------------------+
```

Figure 4: MBAC on the User Level

The user level MBAC is shown in Figure 4. The ADC unit consists of an array of function pointers, adc_algorithm[], with one entry for each CL class. This design, again, allows for the flexibility of using a different admission control algorithms for each CL classes. The ADC Control unit is the function adc_changealgor, through which network administrators can select the admission control routine to use. The third unit, est_changeparam, is the Estimator Control unit for changing the estimation mechanism inside the kernel; this enables network administrators to tailor the averaging algorithm according to their specific needs.

Function Prototypes

We provide a list of prototypes of the major proposed functions and a brief description of each function. Note that the ClassID argument tells each of these functions which CL class to operate on.

Measurement unit:

unsigned int meas_updateCLMq(ClassID *cid, unsigned long packet_size, unsigned int options); meas_updateCLMq updates the bits_sent member of a CLMq entry indexed by *cid.

unsigned int meas_newCLM(ClassID *cid, char *options); meas_newCLM creates a new entry in the CLMq and initialize the storage for *cid.

CLM_entry *meas_readresetCLM(ClassID *cid, CLM_entry *resetvalue, unsigned int options); meas_readresetCLM reads or resets an entry in the CLMq indexed by cid. If the resetvalue is a null pointer, the function will read the entry indexed by *class and return it; otherwise the entry is set to *resetvalue, and the final entry is returned to the caller.

Estimator unit:

unsigned int est_changefp(ClassID *cid, unsigned int EstID, unsigned int options); est_changefp changes the estimation functions according to EstID for class *cid.

ADC unit:

unsigned int *adc_algorithm[](ClassID *cid, unsigned long flowBW, unsigned int options); adc_algorithm takes the class of the new flow *cid in this case) and the flow's bandwidth requirement. It returns a nonzero value if the flow can be admitted and 0 otherwise.

ADC Control unit:

unsigned int adc_changealgor(ClassID *cid, unsigned int AlgorID, unsigned int options); adc_changealgor changes the admission control algorithm of a particular class to the algorithm designated by AlgorID.

Estimator Control unit:

unsigned int est_changeparam(ClassID *cid, unsigned int EstID, unsigned int options); est_changeparam is the user level equivalent of est_changefp in the kernel.

Security Considerations

Security considerations are not discussed in this memo.

References

[B+96] R. Braden (ed.) et al. "Resource ReSerVation Protocol", Internet Draft, June 1996.

[CKT96] C. Casetti, J. Kurose, and D. Towsley. "An Adaptive Algorithm for Measurement-based Admission Control in Integrated Services Packet

Networks", Proc. of the Protocols for High Speed Networks Workshop, Oct. 1996.

[Flo96] S. Floyd. "Comments on Measurement-based Admissions Control for Controlled-Load Service", submitted for publication, 1996. Also available as ftp://ftp.ee.lbl.gov/papers/admit.ps.Z.

[GK97] R.J. Gibbens and F.P. Kelly, "Measurement-Based Connection Admission Control", Proc. of the International Teletraffic Congress 15, Jun. 1997.

[JSD97] S. Jamin, S.J. Shenker, and P.B. Danzig, "Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service", Proc. of IEEE Infocom 97, Apr. 1997. Also available as http://netweb.usc.edu/jamin/admctl/info97.ps.gz.

[JS97] S. Jamin, S.J. Shenker, "Measurement-based Admission Control Algorithms for Controlled-Load Service: A Structural Examination", Univ. of Michigan, CSE-TR-333-97, Apr. 1997. Available as http://irl.eecs.umich.edu/jamin/papers/mbac/clmbac.ps.gz

[Wro95] J. Wroclawski. "Specification of Controlled-Load Network Element Service", Internet Draft, November 1995, <draft-ietf-intserv-ctrl-load-svc-01.txt>.

Author's Address:

Sugih Jamin
University of Michigan
CSE/EECS
1301 Beal Ave.
Ann Arbor, MI 48109-2122
jamin@eecs.umich.edu
+1 313 763 1583
+1 313 763 1503 (FAX)

Cheng Jin
University of Michigan
CSE/EECS
1301 Beal Ave.
Ann Arbor, MI 48109-2122
chengjin@eecs.umich.edu
+1 313 763 6131

Lee Breslau
Xerox PARC

3333 Coyote Hill Road
Palo Alto, CA 94304-1314
breslau@parc.xerox.com
+1 415 812 4402
+1 415 812 4471 (FAX)