ATM Admission Control based on Measurements and Reservations

Herbert Bos
email: hjb1005@cl.cam.ac.uk, phone: +44-1223-334650, fax: +44-1223-334678
University of Cambridge, Computer Laboratory, Cambridge, CB2 3DQ, United Kingdom

Abstract— **We propose an innovative control architecture for ATM that allows users to reserve in advance complex connection patterns in the network. It offers control over the resources via schedules but alleviates the potential over-conservative nature of the resource allocation by coupling it with real-time measurement of actual use of the resources.**

Keywords— **ATM admission control, effective bandwidth, measurements, reservations**

I. Introduction

Size, load balancing and natural distribution often cause sources to be distributed across networks. For example, HDTV video requires 17.5MBps[1] a few hours of which will not fit completely on most storage devices and may well be segmented and distributed. Load-balancing may be a more important reason for distributing data. In [3] a policy is proposed that balances load by chopping up CM files in segments, which are dynamically distributed over the file servers, depending on the load. Furthermore, audio, video and other types of data may be stored on specialised servers for efficiency [10]. Finally, many sources are distributed by nature. Camera's and microphones for example, are attached to workstations or, in the case of security camera's, distributed over a site.

Source distribution becomes a problem if acceptance of a call to one source depends on the acceptance of calls to a set of other sources. We call this *temporal connection dependency*. For example, if a video is distributed over 4 nodes (figure 1), the playback of the entire video requires that segment 1 is played first, immediately followed by segment 2, etc. It is not acceptable that the first 3 connections are accepted while the last one is rejected. It is also not acceptable to set up 4 connections in advance for the entire duration of the file. Instead, it is required that the CAC guarantees that if the connection to source 1 in a sequence is accepted, the connections to all subsequent sources are also accepted. This is required in all systems with temporal connection dependencies. A live example might be a conference where speakers are allocated time in advance[2].

This paper introduces a control architecture (CA) capable of meeting these requirements for arbitrary connection patterns. It then discusses an admission control which prevents over-conservativeness in resource allocation resulting from peak-based reservation. The angle is practical: we describe experiences with an actual implementation.

Section II contains background and related work. Section III discusses the CA. Section IV describes the call admission control, results are presented in section V and section VI draws conclusions.

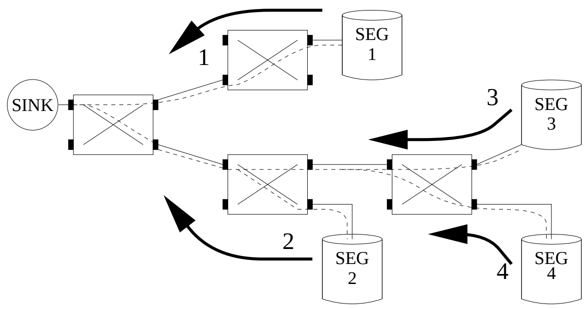

Fig. 1. One CM file consisting of 4 segments

II. Background

Much research is devoted to resource reservation on the datapath. RSVP, for example, offers receiver-initiated reservation. In ATM a variety of CAC algorithms for resource management has been proposed, e.g.: Gaussian Approximation [5], Equivalent Capacity[5], etc.

These algorithms all suffer from relying on a (static) model of the traffic, while often it is impossible to ac-

[1] CBR encoded

[2] As to whether advance reservations are needed at all, we follow [4]: this depends on future scarcity of resources. Where resources are plentiful, not even immediate reservations are necessary. If resources are scarce enough to justify immediate reservations, advance reservations make sense as well

curately characterise sources. An approach that is very often taken in existing policies is the division of calls in a small set of QoS classes. This technique is problematic since one can never define classes that fit all possible types of traffic (including those of future services). Finally, existing CAC also differs from the policy proposed here in that they deal with whether a call can be accepted *now* rather than at some time in the future, which is needed for the guarantees mentioned above.

Recently there has emerged a promising trend to use on-line measurements to overcome some of these problems, e.g. [2] and [7]. In [7] we find the mathematics for a CAC policy similar to the one described here. Several schemes are described and a method is discussed for assigning priorities to traffic classes. No detailed source characterisation is required.

Independent research [4] (based on [6]) addresses similar problems as addressed in this paper for IP based networks. It offers advance reservations and measurements-based admission control. It should be stressed however, that both the network technology and the measurement-based admission control are quite different. Also, the advance reservations for predictive service in [4] are for relatively straightforward flows from A to B only. We shall see that we are able to make a reservation for arbitrary "connection patterns'. It's also different in its objectives[3]. The main difference, however, is probably the underlying measurement-based admission criteria of [6]. The equivalent bandwidth is estimated in rather an ad-hoc fashion. For a fixed number of periods the link utilisation is measured and the equivalent bandwidth is taken to be the *maximum* of the observed rate in each of these periods. This is a simple but crude estimation indeed.

Related work on control of ATM networks is carried out in the OPENET [1] and the xbind projects [8]. Xbind describes a framework for multimedia services on ATM and allows the binding of networking resources to create distributed services. Unfortunately, the CAC relies on a static model of the traffic.

Finally, we should mention that the control architecture builds on previous work in the DCAN project described in [11] where a control architecture called the "Hollowman" is discussed, the components of which serve as a model for the control architecture described here.

[3]It is focused on bounding delay, while we try to manage bandwidth according a desired CLR

III. Control architecture

Guarantees about availability of resources require the control of all relevant resources in the system. We therefore associate a Local Resource Manager (LRM) with a small set of resources, e.g. those on its own host[4] and some dumb devices. The LRM keeps reservations for these resources in allocation schedules. The schedules make it possible to reserve bandwidth of a shared resource in advance, for a specific time interval.

The control architecture (CA) also controls the ATM network. This includes setting up and tearing down connections. For example, if a client C wants to watch the video file of section I, the system has to set up a connection from server 1 to C during time interval $[t_0,\ t_0 + length(seg_1)]$ and from file server 2 to C during $[t_0 + length(seg_1),\ t_0 + length(seg_1) + length(seg_2)]$, etc. We call this a *session*. The CA enables one to *book* connections for a time interval in the future using a simple interface. If all bookings that make up a session succeed, the client can be sure that in the absence of failures it will be able to have all the required connections at the appropriate times. In a session, the architecture reuses existing connections as well as possible, employing something which may be called *multi-source* connections that will accept data from a number of sources in sequence[5].

A. Multi-source connections

The CA employs a new type of connection to connect multiple sources to one sink in sequence. These are not separate connections. Instead, there is one connection that is *time-shared* by several sources. It behaves like a rattlesnake, with its head at the sink and its tail at one of the sources: only the tail moves when the source changes, the rest of its body remains unchanged. The first part of the connection is not even aware of the hand-off of the sources.

For example, in figure 2 a client (sink) requests the playback of a video which consists of two segments on two different file servers. The file servers are connected to 2 switches that connect to the same child switch. The paths from sources to sink have an arbitrary number of switches in common (indicated by the clouds). At the top the connection is from source 1 to the sink, while at the bottom we see that the source has changed from 1 to 2. For the connection,

[4]CPU, disk, network adapter, etc.

[5]This is quite different from such multi-point solutions as in OPENET [1], where multiple sources can send on a distribution tree so that a special adaptation layer is needed.

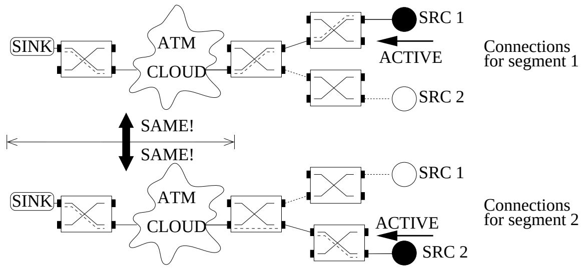

Fig. 2. Rattlesnake: only source moves

however, hardly anything has changed. Only in the very last switch connections the change has been made from source 1 to source 2. The first advantage of such a connection is that we don't have to go through the time-consuming process of setting up the entire connection from source to sink each time the source moves. The second advantage is that we do not waste resources by having multiple connections to the same sink in parallel. And the third advantage is that the sink need not be aware of the handoff at all. The rattle-snake becomes a multi-headed dragon if there is more than 1 sink active (see section III-C).

B. The components of the CA

We describe the various entities in the CA and their interactions (see also [11]). The components of the CA are built on a Corba implementation called DIMMA [9]. In the CA, we have the following entities (see also figure 3): *Host Manager*, *Local Trader, Connection Manager* and *Global Trader*. The Host Manager provides an interface to the CA. Generally, there will be a Local Trader and a Host Manager associated with each host that wants to communicate (possibly running on the same machine).

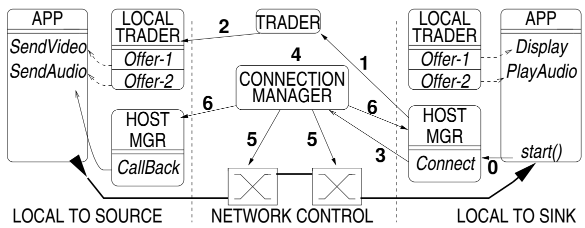

Fig. 3. Interactions between the various components

A typical interaction between these entities is[6]:

- Before we start, the source exports a service offer, which is registered with its Local Trader, e.g. an offer to send video. With the offer the trader registers a callback function. This is the operation (e.g. "send") that will be executed when a client binds to the offer. The sink also registers an offer with its Local Trader, e.g. an offer to display video that is received from the network.
- When a client wants to access this service, it first tries to find an offer for it using its Local Trader, and if needed, the Global Trader, which in turn will talk to the other Local Traders until an offer has been obtained (1 and 2).
- The client also obtains a handle on its local sink offer (from the local trader).
- The client, via the Host Manager, requests the Connection Manager to connect this source offer to this sink offer for this time interval (3).
- The Connection Manager tries to reserve the appropriate resources on the data-path and if this completes successfully, it sends back an acknowledgment to the Host Manager.
- The request then sleeps (4) until t_{start}, at which point it sets up the connection (5).
- If successful, the Connection Manager tells the Host Managers on both sides to kick the callback operations (6): the source is told to send video and the sink is told to receive and display frames.

[6]described is a connection setup between source and sink program; this is by no means the only mode of operation of the CA, but here it is the most relevant one.

C. Interfaces

The CA offers an interface that allows users to do a wide variety of things. We will discuss only a small subset of the CA's operations that is relevant for systems with temporal connection dependencies.

Everything is *timed* in the CA. Instead of setting up a connection between A and B, the user asks the CA to set up a connection from A to B, starting at time t_{start} and ending at time t_{end}[7]. The operation for setting up a timed connection from source offer to sink offer is (`pflag` indicates whether connections will be persistent)[8]:

```
connectSrcToSink (Offer src, Offer sink, Bool pflag,
                  Bandwidth peak, Time t1, Time t2);
```

A more advanced way of accessing the CA introduces the concept of *connection sequence*, which is a sink offer together with a list of source offers and times. Each item in the list contains a source offer as well as an end time $t_{end}(i)$ for this source. For

[7]A traditional connection has interval $[now, \infty\rangle$.

[8]Although it seems that the only QoS dealt with here is bandwidth, it is observed that QoS parameters are not orthogonal and that it is possible to map parameters such as delay, delay variance, etc. on the service rate[2]. We will assume a similar mapping and not discuss other QoS parameters in great detail.

each item i in the list, the control architecture connects the source offer to the sink offer until the time is $t_{end}(i)$, at which point connects the sink to the next source offer. The establishment of a connection sequence implements the rattle-snake connections described in (III-A). The operation is as follows:

```
connectSequence (List<OfferAtTime> reqsForSrcOffers,
                 Offer aSinkOffer, Time start);
```

where `OfferAtTime` is a type that corresponds to an item in the list. An element of this type contains a source service-offer reference, the peak rate for this source, as well as an end time, as follows:

```
class OfferAtTime { Offer src; Bandwidth p, Time t;}
```

The final step is to extend the functionality of the connection sequence (which provides multiple sources) by adding multiple sinks to it. We call this a *connection pattern*. It allows users to specify a sequence of sources with corresponding non-overlapping intervals and associate with these a set of sinks. The sinks are also tied to specific intervals but these *are allowed* to overlap (and may span several sources).

Overlapping sink intervals indicate multi-casts: the data from the active source will be sent to all sinks that overlap with the source interval. In the CA, multiple sinks connect to an active source that already has a sink attached to it by executing a `join` operation[9]. The connection-pattern operation is as follows:

```
connectPattern(List<OfferInInterval> sources,
               List<OfferInInterval> snks, Bool pFlag);
```

where `OfferInInterval` is similar to `OfferAtTime`, except that an interval instead of a single time is associated with the service offer:

```
class OfferInInterval { Offer anOffer; Bandwidth peak,
                        Time start;    Time end; }
```

Replacing sources in multicast trees

This last operation introduces an interesting problem regarding the activation of a new source. In figure 4, SRC1 acts as the source for a multicast tree. At some point SRC2 has to take over from SRC1. Instead of setting up a whole new tree from SRC2 to all sinks (which yields an *optimal* tree but suffers in performance because much of the existing tree *could* be reused), we adopted the following solution. We find the first node that the multicast tree for SRC2 would have in common with the old multicast tree, the first common node (FCN). We simply reuse the tree underneath it. From the FCN we then establish a connection to the root and reuse all dangling subtrees along the way (figure 5).

[9]called from within the control architecture and transparent to user; they simply bind to the source offer

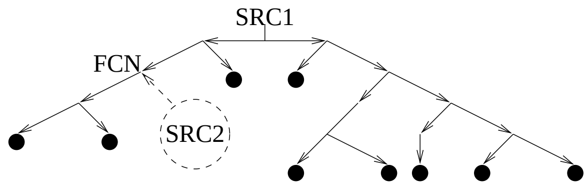

Fig. 4. A new source takes over

```
up = down = FCN = oldTree.firstCommonNode(newSource);
// if a subTree of the old tree exists at node FCN
if (oldTree.existsSubTreeAt (FCN)) {
   tree_t branch = oldTree.unconnectSubTreeAt (FCN);
   newTree.appendSubTreeAt (branch, FCN);
}
while (down != root) {
   up.goUp();
   // now reverse the direction of the connection
   newTree.createConnection (down, up);
   oldTree.releaseConnection (up, down);
   // if a subTree of the old tree exists at node up
   if (oldTree.existsSubTreeAt (up)) {
      tree_t branch = oldTree.unconnectSubTreeAt (up);
      newTree.appendSubTreeAt (branch, up);
    }
    down = up;
}
```

Fig. 5. Source replacement algorithm

IV. Call Admission Control

A basic CAC algorithm would simply check the reservation schedules to see if enough bandwidth is available to accept the new request and if so it would update the schedules. Depending on how requests are entered in the schedules, this may result in very poor resource utilisation. If, for example, the requests are entered based on peak rates, the result would be extremely conservative[10]. Note that statistical multiplexing for a time interval in the future is difficult due to the unknown behaviour of future flows.

Many CAC algorithms have been proposed in the literature. In section II we mentioned that a major problem of most of these existing methods is their relying on a static model of traffic (in other words, accurate source characterisation). We propose CAC based on schedules and real-time traffic measurements that has no need for such a model. Bear in mind that the effective bandwidth (EBW) discussed here is the buffer service rate required to keep the cell loss ratio (CLR)[11] under a target bound.

A. Proposed CAC policy

The CAC algorithm is based on the traffic patterns observed during a time interval in the past. These patterns are used to estimate the traffic that will be generated if we include the new request. If this does not exceed the capacity, the call can be accepted. The effective bandwidth used for this is always less than (or equal to) the peak rate. Therefore, CAC using EBW gives better resource-utilisation than CAC based on

[10]but guarantees would be relatively hard

[11]due to buffer overflow

schedules alone[12]. At first sight, it seems that measuring traffic is unsuitable for our CA, because requests are made for an interval in the future: at admission time there is nothing to measure. On the other hand, we do have knowledge about the behaviour of current streams. This can be used for CAC in the near future.

The CAC in the CA is an attempt to bring together the strictness of schedules and the resource utilisation resulting from measurements. It may be thought of as a sliding scale with *measurements* at one extreme (corresponding to $t = now$) and schedules at the other (corresponding to $t = \infty$). Starting with CAC based on measurements and looking further into the future, we will see that the CA's CAC grows progressively more conservative (because we know nothing about the future streams, except their peaks). As time goes by we learn more about the flows in a specific interval that were admitted with the conservative CAC (some reserved calls will have started), so we can now make less conservative decisions for new requests for that interval.

B. Implementation

If we just consider schedules, the way the CAC might work when a new request comes in is as follows:
1. Let B_{total} be the total bandwidth on the resource
2. For the time interval that is specified for the new request, determine $B_{scheduled}$, the maximum amount of bandwidth reserved by adding all the peak rates of the reservations
3. Let b_{new} be the new request's peak rate
4. If $B_{scheduled} + b_{new} \leq B_{total}$ accept the request. If not, reject.

This is one of the most conservative schemes possible that works on a first-come-first-serve basis. The resource utilisation with this CAC will be low, but the resource guarantees are relatively hard. Next, we add measuring components to the resources. For switch ports, we add code that periodically sends the cell count per active (vpi, vci) to a *traffic server* which computes estimates for the effective bandwidth (EBW) of each of the connections (corresponding to a target CLR). It also obtains an estimate for the *aggregate* EBW.

In the schedules, we keep track of the EBW of the flows. Initially, the EBW is set to the peak rate. So, if the reservation is for time interval $[t_0, t_1]$ and $now \leq t_0$, we use the peak rate for CAC. For each new request, the CAC asks the relevant traffic servers for the EBWs of the active connections and also for the aggregate EBW.

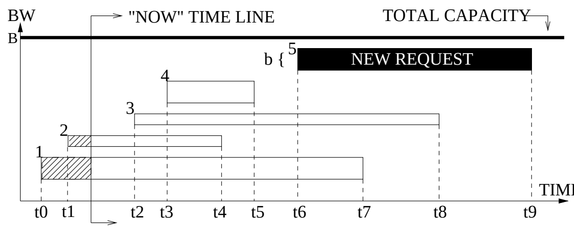

Fig. 6. A new reservation request arrives

See figure 6. At $t = now$ a request comes in for bandwidth in $[t_6, t_9]$. The resource is a switch port. At $t = now$ there are 2 active connections (1, 2) of which we have EBW estimates. Connection 2 finishes *before* the new request starts, while the other 1 overlaps with the new reservation's interval. Between *now* and t_6 2 more connections are set up (3 and 4) one of which finishes before t_6. Finally, we obtain the aggregate EBW. The CAC algorithm is now:

1. Let B_{eff} be the aggregate EBW that is used on the resource, $b_{eff}(n)$ the EBW of connection n and $peak(n)$ the peak rate of connection n
2. At $t = now$ a new reservation request r comes in for interval $[t_{start}, t_{end}]$ with $peak(r) = b$.
3. Let E_{BW} be the estimated maximum of the bandwidth in $[t_{start}, t_{end}]$ (i.e. the bandwidth that we use to decide whether the request should be accepted or not). Initialise E_{BW} to B_{eff}.
4. For all active flows x that finish *before* t_{start} do: $E_{BW} = E_{BW} - b_{eff}(x)$.
5. For all reservations y of which the connections have not started yet and which *overlap* with $[t_{start}, t_{end}]$, do: $E_{BW} = E_{BW} + peak(y)$.
6. If $(E_{BW} + b \leq B_{total})$ accept, otherwise reject.

So for reservations in the future we use the specified peak rates, while for connections that exist already we can use the effective bandwidth estimated by the measure algorithm in the traffic server. Further in the future the CA will be more conservative in its admission control policy.

Traffic servers

Traffic servers are independent processes which talk to the switch on one end and the control architecture on the other. They receive raw statistics from the switch, process them and send updates of the EBW to the CA (on request). The traffic server is

[12] the EBW estimators were developed as part of the *Measure* project, and used some code written by Horst Meyerdierks

optional[13]. If a switch cannot provide the statistics, the CA still works (albeit more conservatively). In fact, it is conceivable to control a heterogeneous network where some switches (or indeed some of the ports of these switches) have traffic servers, while others do not. Also, the implementation may vary from switch to switch, allowing vendors to differentiate.

Effective bandwidth estimation

Our CAC algorithm is based on [2]with few modifications. We briefly discuss some of the results. Assume our buffer is served at constant rate r. The workload process $W_t = \sum_{i=1}^{t} X_i - t \times r$ [14]. The queue length Q depends on W_t: $Q = max\{W_t : t \geq 0\}$. Under general conditions, a single-server queue has a queue length distribution with asymptotes of the form:

$$P(W_t > x) \asymp e^{-tI(x)} \quad \Rightarrow P(Q > q) \asymp e^{-\delta q} \tag{1}$$

$I(x)$ is the rate function of the workload process and δ is called the decay-rate. Now, the r that corresponds to a CLR can be calculated as follows.

Define λ, a transform of I called the *scaled cumulant generating function* (SCGF), of the workload process as follows:

$$\lambda(s) = \lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{sW_t}\right),$$

related to I by the Legendre transform:

$$I(x) = max_s\{xs - \lambda(s)\} \tag{2}$$

$r \times t$ is constant, so $W_t = \sum_{i=1}^{t} X_i - r \times t$, so:

$$\lambda(s) = \lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{s(\sum X - r\times t)}\right) =$$
$$\lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{s(\sum X)}\right) - s \times r = \lambda_A(s) - s \times r$$

where λ_A is the SCGF of the arrivals process. So, given the arrivals SCGF we can calculate δ as function of r as follows:

$$\delta(r) = max\{s : \lambda_A(s) \leq s \times r\} \tag{3}$$

We use this to find r, the service rate corresponding to a particular δ, i.e. to the target CLR. Using(1):

$$P(Q > q) \asymp e^{-\delta(r)q} \Rightarrow r = min(\delta(s) \geq -\frac{\ln CLR}{q})$$

and from (3) we know that $r \geq \lambda(\delta)/\delta$. Substitute $\delta = -\frac{\ln CLR}{q}$, so: $r = \lambda\left(-q^{-1}\ln CLR\right)\left(q^{-1}\ln CLR\right)^{-1}$ and since X_n iid:

$$\lambda(-x) = \lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{-x(\sum(X_i) - xt)}\right) =$$
$$\lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{-x\sum(X_i)}\right) + xt =$$
$$-\lim_{t\to\infty} \frac{1}{t} \ln E\left(e^{x\sum(X_i)}\right) + xt = -\lambda(x)$$
$$\rightarrow \text{so we find}: \ r = \left(\lambda\left((\ln p)/q\right)\right)/\left((\ln p)/q\right) \tag{4}$$

The function $\lambda(\theta)/\theta$ is called the effective bandwidth. To estimate λ we use the following: for a large class of arrival processes it is possible to find a block length T such that the aggregated arrivals A_T are approximately iid. Then:

$$\lambda(s) \approx \frac{1}{T} \ln E\left(e^{sA_T}\right), \text{so } \hat{\lambda}(s) = \frac{1}{T} \ln \frac{1}{N} \sum_{i=1}^{N} e^{sA_T^{(i)}} \tag{5}$$

gives an estimation of the EBW. T should be large enough to make arrivals in that interval independent, but not so large that short bursts are smoothed out[15].

V. Performance

We test the CA with a focus on the CAC algorithm. The ATM test-bed contains a number of Fore switches attached to HP, DEC Alpha and Solaris machines as well as some ATM cameras. Communication in the CA is based on IIOP or ANSA rpc.

Connection setup

Making a point-to-point reservation using unoptimised code takes 30 ms on a DEC Alpha. The overhead incurred by setting up the call at wake-up time is null when compared to a CA without advance reservation[16]. So for an end-to-end connection across 2 switches (including callback) we measured 322 ms for immediate setup and 301 ms for the connection setup resulting from advance reservation. The reason why it is so slow is twofold: a slow implementation of IIOP[17] and the use of SNMP for communication with one of the switches (GSMP is used for the 2nd switch). SNMP constitutes the bulk of the overhead. SNMP connection setup takes hundreds of milliseconds, while with GSMP 6-8 ms was achieved [12], so we conclude

[13] and can be replaced at runtime

[14] $\{X_i\}$ is the number of arrivals in interval i (iid)

[15] we have used $T = 200ms$

[16] in fact, connection setup itself is now even faster, because initial communication and CAC overhead was already absorbed when the reservation was made

[17] Each IIOP invocation takes 5 ms; these days latencies of 1-2 ms are achievable for commercial implementations

that SNMP is a bad choice for open signalling[18]. In [12] a break-down of the overhead suggests that a setup time across a switch of approximately 10 ms is achievable.

Admission Control

We submit a set of requests to the CA. The resource is a switch port shared by all calls. The relevant parameters are shown in figure 7. Note that the sum of the peak rates of source 1 (5000 cells/s) and source 2 (4000 cells/s) exceeds the capacity of the port (8000 cells/s). At $t = 0$, we try to reserve bandwidth for 3 calls, but since the CAC is based on peak rates only (no measurements yet), the request for source 2 (which overlaps with source 1) is rejected. Figure 8 shows the actual traffic of the flows[19].

TIME (s)	RESERVATION	INTERVAL	CAC RESULT
0	SOURCE 1	[1, 300]	ACCEPT
	SOURCE 2	[150, 300]	REJECT (!)
	SOURCE 3	[300, 450]	ACCEPT
50	SOURCE 2	[150, 300]	ACCEPT (!)

Resource capacity: 8000
Peak (source 1): 5000
Peak (source 2): 4000
Peak (source 3): 7000

Fig. 7. The requests, peak rates and capacity

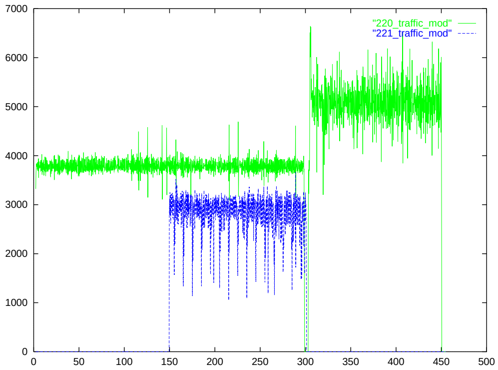

Fig. 8. Traffic on connections

The total traffic on the port is shown in figure 9. Also shown is the aggregate effective bandwidth which lies well below the peak rates. Since our CAC uses the EBW of active calls rather than their peaks, it makes a less conservative admission decision after the first connection starts. So we see (figure 7) that when we try to reserve for source 2 again at $t = 50$, the request is now accepted. The EBW of the active connection (< 4000) plus the peak of the new request (4000) does not exceed the resource capacity. Figure 9 shows that the acceptance is justified—the total traffic never exceeds the capacity and resource utilisation improves considerably.

[18] the reason for using it at all is that all switches support it

[19] Note that although there are 3 connections, we only use 2 vci values. Since the third connection starts after the first connection has ended, it reuses its vci (220)

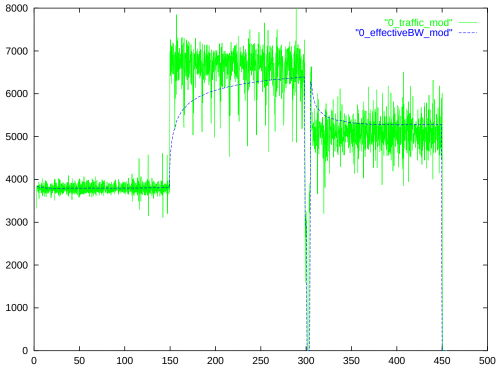

Fig. 9. Total traffic and effective bandwidth

VI. Conclusion

We have described a control architecture (CA) that allows for advance reservation of resources. Its admission policy merges traffic measurements with rigid scheduling. This results in good resource utilisation without the (almost impossible) requirement of accurate source characterisation.

References

[1] I. Cidon, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M. Sidi, *The OpeNet Architecture.* Sun Microsystems Technical Report 95-37, December 1995

[2] S. Crosby, J.T. Lewis, I. Leslie, N. O'Connell, R. Russel, F. Toomey, *Bypassing Modelling: an Investigation of Entropy as a Traffic Descriptor in the Fairisle ATM Network.* Proceedings of 1st Workshop on ATM Traffic Management, 1995

[3] A. Dan, M. Kienzle, D. Sitaram, *A Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand servers.* IEEE Multimedia, 1995

[4] M. Degermark, T. Kohler, S. Pink, O. Schelen, *Advance Reservations for Predictive Service.* Proceedings of NOSSDAV'95, pp 3-14, April 1995

[5] R. Guerin, H. Ahmadi, M. Naghshineh, *Equivalent Capacity and its Application to Bandwidth Allocation in High-Speed Networks.* JSAC, Sept 1991

[6] S. Jamin, P. Danzig, S. Shenker, L. Zhang, *A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks.* Proceedings ACM SIGCOMM'95, 1995

[7] R.J. Gibbens, F.P. Kelly, *Measurement-based connection admission control.* 15th International Teletraffic Congress Proceedings, June 1997

[8] A.A. Lazar, K.S. Lim, F. Marconcini, *Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture.* JSAC, Sept 1996

[9] G. Li, *DIMMA Nucleus Design.* APM Technical Report 1553.00.05, October 1995

[10] S.L. Lo, *A Modular and Extensible Network Storage Architecture.* University of Cambridge, Technical Report No. 326 (Ph.D. thesis), January 1994

[11] S. Rooney, *The Hollowman: An Innovative ATM Control Architecture.* Proceedings of IM'97, May 1997.

[12] S. Rooney, *Connection Closures, Adding Application-defined behaviour to network connections.* ACM SIGCOMM, Comp. Communication Review, April 1997