On Large Fan-Out Multicasting

Dimitrios Stiliadis
Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07747
e-mail: stiliadi@bell-labs.com

Abstract—Broadband access technologies such as DSL, cable modems, Passive Optical Networks (PON) and Fiber-to-The-Home are breaking the last mile bottleneck, enabling applications that distribute multimedia content over data networks. Several of these applications, like broadcast video and interactive gaming, require multicast distribution of content. An important problem, from the system architecture perspective, is to distribute multicast content over a large number users (i.e. large fan-out multicast) by maintaining low system cost and without deteriorating system performance. In order to forward multicast packets, the system must replicate the same packet, or pointers of the packet, to multiple interfaces. This function requires significant speed-up from the memories and the forwarding engines. Previous efforts in solving this problem have only led to statistical solutions, that cannot provide deterministic performance. As a result, temporary overload in multicast traffic could lead to unnecessary packet losses, even for unicast connections that share the same interfaces. In this paper, we present a novel algorithm for managing multicast sessions with large fan-out, that offers isolation between interfaces, deterministic performance, and flexibility in the offered delays. To the best of our knowledge, this is the first algorithm ever proposed that can provide deterministic performance for multicast sessions.

Index Terms—Multicast, hardware, delay bounds, deterministic analysis.

Methods Keywords: system design, deterministic network calculus

I. INTRODUCTION

Access technologies such as DSL, cable modems, and Passive Optical Networks (PON) provide high bandwidth access to end users. One of the key services that providers would like to offer over broadband access is multimedia services such as broadcast digital video, video on demand, teleconferencing and interactive gaming. An important component for enabling such applications is support of efficient multicasting in edge switches and routers.

For example, broadcast video is delivered to the edge of the network over a satellite links. The digitized video is then distributed over the access packet network to end customers. Edge switches and routers are serving several hundreds of end-users and they must be able to efficiently distribute the multicast content at low cost. A general model of the architecture of such a switch is shown in Figure ???. In general an edge router consists of a set of a line cards, which are inter-connected through some form of switching network\(^1\). Some of the line cards provide the uplink to the backbone and are referred as trunks. The rest of the line cards support a large number of access interfaces, such as DSL links, PON, or channelized SONET, that provide point-to-point connections to the end users. Each switch fabric port has capacity \(C\) and each of the access links has capacity \(C_i\).

Packets that arrive from the trunk line cards can be multicasted to the access line cards using one of several multicast enabled switching fabrics \([\text{?}], \text{[?]}\), \([\text{?}], \text{[?]}\), \([\text{?}], \text{[?]}\), \([\text{?}]\). It is common practice for some of the buffering of the system to be provided at the egress line card. Note, that output buffering is definitely a requirement in switch fabrics based on output queued architectures, as well as Combined Input Output Queued (CIBQ) architectures \([\text{?}]\). Output buffering can also become a requirement in input-queued fabrics, when line cards support low speed access interfaces. This is due to the capacity mis-match between fabric ports and access links. Large bursts of data might arrive from the switch fabric with rate \(C\) destined to one of the access links, that can only drain the data with rate \(C_i << C\).

The egress logic of the line card, commonly referred as the traffic manager, receives only one instance of the multicast packet and is responsible for replicating, queueing and delivering packets to the access interfaces. The number of interfaces that packets of a multicast session must be replicated to is referred as the fan-out of the session. In this paper, we concentrate on systems that must support high fan-out multicast sessions, or high-speed interfaces. Note, that although we present the problem by referring

\(^1\)We show the ingress and egress functions of the line cards as separate logical blocks, although they are physically located on the same card.
to access routers, the same problem must be solved in the case of a multicast network overlayed over a unicast core network. In such a network, not all routers are multicast enabled, and multiple virtual interfaces must be supported over the same physical interface [1]. Multicasting in this context, corresponds to transmitting the same packet multiple times over each physical interface (once for each virtual interface). This mode of operation might also result in large fan-out multicasting.

Replicating packets, either through pointer manipulation or actual copying can be a very expensive operation both in terms of processing cycles and/or memory bandwidth [2]. The multicasting process must copy each packet to a large number of queues, and this operation must be completed at “wire-speed” (i.e. assuming a continuous stream of packets of the smallest size, the multicasting process must be completed without delay) [2]. This requirement translates to a speed-up requirement from the memory that can be of the order of \( O(N) \) where \( N \) is the number of access interfaces. In other words, if the system supports only unicast traffic, the memory bandwidth needed is of the order of \( O(C + \sum_{i=1}^{N} C_i) \). For arbitrarily large fan-out multicast traffic, the memory bandwidth required is increasing to \( O(NC + \sum_{i=1}^{N} C_i) \). Note, that the problem is very similar to that of multicasting in shared-memory packet switches [2]. However, previous work on this subject has only provided solutions that can work in a statistical sense for some arrival traffic patterns, and cannot provide deterministic performance.

The main contribution of this paper is the design of algorithms for replicating packets across a large number of interfaces that do not require any speed-up from the memories and provide deterministic performance to multicast sessions. In detail, the algorithms proposed allow to:

1) Simplify the multicast operation and allow arbitrary fan-out multicast without any speed-up requirements from the traffic manager.
2) Offer finite bounds on memory requirements.
3) Provide deterministic bounds on the latency of multicast connections during the replication process.
4) Allow the support of layered multicasting where different interfaces might receive a different number of layers of a multicast application, depending on the available bandwidth, without requiring the establishment of multiple multicast groups.

The rest of the paper is organized as follows: In Section ?? we provide an overview of the system model and some background on different multicast replication algorithms. In Section ?? we describe a multicast replication algorithm and prove its stability. In Section ?? we extend the model to a range of algorithms and analyze their properties. This extension allows system designers to peak a multicast replication algorithm from a whole set of algorithms and trade implementation complexity to system performance. In Section ?? we present some simulation results of the proposed scheme, as well as a comparison with previous approaches. Finally, in Section ?? we discuss the implementation of the algorithm.

II. BACKGROUND AND SYSTEM MODEL

Before we discuss the requirements that are imposed by multicasting, we will outline some of the architecture principles of the traffic management component. Assume that the switch fabric delivers packets to the traffic management unit with a peak capacity \( C \) bits/second. Assume also that the capacity of each outgoing interface \( i \) is \( C_i \) bits/second. In systems that provide broadband access it is common to assume that not all interfaces are requesting peak bandwidth at the same time, and the buffers at the egress line card are used to provide some form of statistical multiplexing among the end users, in order to reduce cost. For example, in fiber-to-the-home applications the peak bandwidth to each end-user might be 100 Mbits/second. However, the allowed access bandwidth might be much lower, so that providers can offer different levels of service at different price points. For these reasons, we will not assume any relationship between the capacity of the switch fabric port \( C \) and the capacity of the interfaces \( C_i \).

When designing a traffic management component, even for supporting unicast traffic, one has to keep in mind

\[ \sum_{i=1}^{N} C_i \leq C. \]

Note, that it is common practice in general router design to assume that \( \sum_{i=1}^{N} C_i \leq C \). We emphasize that this is not always the case in edge systems.
two distinct memory bandwidth requirements. First, there must be enough memory bandwidth available to write and read all packets from the memory. Thus, a memory bandwidth of at least \( C + \sum_{i=1}^{N} C_i \), where \( N \) is the number of interfaces is necessary for buffering the packets.

In addition, the traffic management component must process the data structures that maintain the queues. In order to provide some quality-of-service differentiation, the system will support at least a small number of queues for each access interface. For example, support of diff-serv will require up to eight queues per interface [?]. Since we assume that the number of interfaces is large, the total number of queues that must be maintained becomes also large. Maintaining a large number of queues is efficiently done by implementing queues using linked-lists. Keep in mind also, that variable size packets are stored in terms of linked lists of constant size buffers [?]. Therefore, queues are implemented as linked-lists of buffers. Maintaining linked lists is a demanding operation in terms of memory bandwidth, since link list accesses are random in nature and they cannot explore the features of burst-access memories.

In order to support wire-speed operation, the traffic management component must be able to handle arbitrarily long streams of minimum size packets [?]. If we denote with \( P_{\text{min}} \) the length of the minimum size packet, and we assume that all packets are unicast, the bandwidth needed for updating the queue data structures is of the order of \( O\left( \frac{C + \sum_{i=1}^{N} C_i}{P_{\text{min}}} \right) \). We use an order notation, since the exact number of memory accesses for inserting to and deleting a packet from a queue, depends on the buffer management and scheduling architecture, as well as the width of the memory and might vary significantly. As a rule of thumb, readers can assume that at least 4 memory accesses are needed for each insertion and 4 memory accesses for each packet transmission.

When handling multicast packets, any single packet might be added into several queues and/or can be read several times from the memory. Although the actual packet might be stored only once in the memory, several data structures might be updated. There are several simple methods to implement the multicast replication function that we outline below [?].

A. Maximum Speed-up

A packet is stored in the memory only once, and a descriptor of the packet is added to all the interface queues that the packet must be transmitted to. This scheme is also known as Replication-at-Receiving or (RAR). The RAR approach minimizes the memory bandwidth needed for storing packets, although it does not improve on the memory bandwidth and processing requirements for handling the queue data structures. Since multicast groups might be of arbitrarily large fan-out, the traffic management component must have enough memory bandwidth to add a descriptor of each incoming multicast packet to all outgoing queues. This translates to a control memory bandwidth requirement of the order of \( O \left( \frac{N^2+C+\sum_{i=1}^{N} C_i}{P_{\text{min}}} \right) \). In other words, the control memory requires a speed-up of \( N \). Obviously, this becomes prohibitively expensive for large \( N \).

Let us take for example a DSL access router. Let us assume that the switch port capacity is 1.2 Gbps, and each line card supports 128 VDSL lines. Each VDSL line supports bandwidth of 10Mbits/second. Assuming minimum size IP packets (or ATM cells in the case of DSL) of 40 bytes, and unicast only operation, the traffic management unit must be able to insert and delete from the data structures approximately 3.1M packets/second. If multicast with full copying capabilities must be supported, the necessary processing capacity is increasing to 400M packets/second. Obviously, support of arbitrary fan-out multicast becomes very expensive and impractical, even if there are memories available that can provide this type of bandwidth.

The problem with the above design is that the bandwidth required is determined by a worst-case design and it is probably unused most of the time. However, designing without such a speed-up would result in dropping multicast packets and/or sacrificing wire-speed operation. Indeed, if a speed-up of less than \( N \) is available, it is easy to create an arrival traffic pattern where back-to-back packets must be transmitted to a number of interfaces larger than the available speed up. By the time the second packet arrives, the insertion of the first packet into the queues is not complete, and the second packet must be dropped.

B. Staging Buffers

It is exactly the above observation that led to the following alternative architecture presented in Figure ?? [?], [?]. Multicast packets are placed on a small queue on arrival, that is called a staging buffer. The traffic manager replicates the packets to the per-interface queues with some speed-up that is less than the maximum needed. For example, during a packet arrival time, the traffic management unit can place up to two packets in the interface queues; one unicast packet arriving from the fabric and one multicast packet from the staging buffer.

This method might work for several traffic patterns, but does not offer deterministic non-blocking operation. Assume that the rate of arrival of multicast packets is \( r_m \).
Assume that the traffic management unit can process multicast packets with a rate of $\rho_m$ and assume that the average fan-out of the multicast packets is $f$. The average rate of processing required by the traffic management unit is equal to the rate of multicast packet arrivals multiplied by the average fan-out. It can be easily seen that if

$$f \cdot \rho_m \geq \rho_m$$

the system can become unstable. Even if the system is stable based on the average rates, bursts of multicast packets can create short-term instabilities. Since the length of these bursts is not known a-priori, determining the amount of buffers needed to minimize losses in such a system can become a major problem [?].

C. Dedicated Multicast Queues

An alternative approach, also known as Replication-at-Send (RAS), is to queue multicast packets independently in one or more dedicated multicast queues. In this case, packets destined to different interfaces might be buffered together in the same queue. The multicast queues are visited periodically, the first packet is read, and is replicated across the destination interfaces on the fly [?]. Multicast packets are not queued on a per-interface basis.

In the context of access systems, the interfaces can consume packets with a much lower rate than the traffic management unit is processing packets. One can realize that this property can lead to head-of-line blocking. Consider the example of Figure ???. A burst of minimum size packets arrives, where the first 4 packets belong to the same multicast session and are destined to the same interfaces. The fifth packet belongs to a different multicast session and is destined to different interfaces. The system supports 16 interfaces, where the speed of each interface is $\frac{1}{16}$th of the speed of the switch fabric port. The first packet is read from the memory and copied to interfaces 1, 2, 3 and 4. The interfaces remain busy until times $t_{16}, t_{17}, t_{18},$ and $t_{19}$ respectively. However, the traffic manager finishes copying the first packet at time 3. At time 4 it peaks a new multicast packet from the queue that must be also copied to the same interfaces. However, since these interfaces are busy, the traffic manager is forced to stall its operation and wait until they become free again.

There are several heuristics that one might attempt in order to solve the above problem. The traffic manager can look farther into the multicast queue and try to find a packet that will not block its operation. Alternatively, a small buffer can be added before each interface to absorb bursts of packets [?]. Although both of these approaches can improve the performance of the system for several traffic patterns, they do not deterministically solve the problem, and it is always possible to create a long enough burst that will stall the system.

In addition, since packets for different interfaces are queued in a single queue, one can not differentiate between congested and non-congested interfaces and transmit a different percentage of the multicast packets to each access interface depending on its congestion. A congested interface will lead to a long multicast queue and unnecessary packet drops, although other multicast packets can be delivered to other interfaces that are not congested.

It is clear from the above discussion, that previous approaches can provide some statistical improvement on system performance, but they cannot guarantee deterministic operation. More importantly, system design might depend on arbitrary assumptions on arrival traffic patterns, and this can lead to poor performance. In addition, intelligent buffer management and or Quality-of-Service guarantees are impossible with any of the above approaches. In the next sections we will describe and analyze a method that can guarantee deterministic behavior for multicast traffic without any speed-up.
III. BATCH MULTICAST

The scheme we will describe is based on some simple data structures that allow us to batch the work of replicating packets. We use a staging queueing structure, where multicast packets are queued on a per-session basis. In Figure ?? we show a snapshot of the data structures of the system. Note, that per-session state is required anyway for any multicast implementation, since for each multicast session we must maintain a list of interfaces that the packet must be replicated to. The additional state information required by the algorithm is two pointers, that correspond to the head and tail pointers of a queue of packets waiting to be replicated. We will denote the data structures together with the process that replicates packets to interfaces as the replication server. A pseudo-code of the algorithm is presented in Figures ??, ??, and ??.

When a multicast packet arrives it is placed at the end of the queue that corresponds to this multicast session. A ref-
Per Interface Transmission Process

while queue of descriptors is not empty
    Retrieve the first descriptor
    while more packets pointed by descriptor
        Transmit the packet
        Reduce the reference count by one
        if reference count equals 0 then
            release packet buffers
        end if
        go to next packet
    end while
    release descriptor
end while

Fig. 7. Description of functions performed on packet arrival

ference count equal to the fan-out of the multicast session is associated with each packet. If this is the first packet of the session, a descriptor corresponding to this session is added to a list of multicast sessions that have packets waiting to be replicated. We denote this list as the replication set. The replication server processes one session at a time in round-robin order, and can perform only one copy during the time interval that a minimum size packet arrives from the fabric (This time interval is referred as packet-time)\(^3\). We will actually show that the traffic manager can perform less than one copy per time, and as long as the rate of copies is bounded, the system will still be stable. Note, that each group is added to the list of groups waiting to be replicated only once, even if new packets arrive for this session while it is waiting to be replicated.

The replication process is described in Figure ???. In order to copy the packets of a multicast session to an interface queue we create a virtual descriptor that points to the list of packets of this multicast group that have arrived until that time. The descriptor is now added to the corresponding interface queue, one interface at a time. The use of descriptors is necessitated by the fact that multiple sessions might share a single interface queue, and we do not want to impose to the interfaces a per-flow queueing behavior.

Allocating a descriptor and adding it to the interface queue is a constant time operation. However, during this operation a large number of multicast packets might be added at once to any given interface. When the operation of the system starts, the complete replication of the first session will take several packet times, as determined by the fan-out of the session. During this time interval, multiple multicast packets might arrive for this, or other multicast sessions. These packets are queued on the corresponding session queues. When the process replicates packets from the next multicast session, it will replicate a list of packets instead of a single packet. The key reason that the above scheme works is that the replication rate in terms of multicast packets per second increases as the arrival rate increases. However, the replication rate in terms of sessions per second does not need to be increased and no memory speed-up is required.

The per-interface queues are also shown in Figure ???. Each queue is essentially a list of descriptors and each descriptor points to a set of packets. The transmission process visits each descriptor and transmits all packets attached to it. Every time a packet is transmitted, the corresponding reference count is decreased by one. When the reference count reaches zero, it means that the packet has been transmitted to all interfaces in the multicast session, and the memory occupied by the packet can be released. A description of the transmission algorithm is shown in Figure ???.

A. Stability Analysis

We will first discuss the stability properties of the replication server as it was described above. This discussion will provide the intuition that will help us define a whole range of replication systems that will allow us to vary the delays and buffer requirements of different sessions.

We first introduce some notations and definitions:

**Definition 1:** A session is **backlogged** as long as the corresponding session queue is non empty or the packets of this session are currently being replicated.

Without loss of generality we can normalize time to the minimum packet time. We assume that \( f_i \) is the fan-out of session \( i \). We first prove the following Lemma:

**Lemma 1:** The time interval between the time that a session is added to the tail of the replication server and the time that the replication for that session is complete is bounded by

\[
\Theta_i = \sum_{i=1}^{K} f_i
\]

where \( K \) is the number of multicast sessions.

**Proof:** Sessions are served in round-robin order, and any given session is only queued once in the replication list. When the session is added to the tail of the replication list,
all other sessions might have already been queued ahead of it. Each session \( j \) requires time equal to its fan-out \( f_j \) to be replicated. Thus, session \( i \) will be served at the latest after time \( \Theta_i \).

We can now prove the stability of the replication system:

**Lemma 2:** If the replication server can guarantee that any session will be visited within \( \Theta_i \) time after it is added to the tail of the replication queue, the delay of any packet in the replication stage is bounded by \( \Theta_i + f_i \).

**Proof:** Assume some multicast packet arriving for session \( i \) at some time \( \tau \). If this is the first packet for this multicast session, the session is added to the list of sessions that must be served. Based on Lemma 2, the server can guarantee that a session will be visited at the worst-case after time \( \Theta_i \) from the time it is added to the replication list. Thus, the packet will be served the latest by time \( \tau + \Theta_i \). If this is not the first packet for this session, then some other packet that belongs to this session has arrived at some time \( \tilde{\tau} < \tau \). In this case, session \( i \) is either waiting to be served, or it is in the process of being served. If the session is waiting to be served, then all packets of the session will be served by time \( \tilde{\tau} + \Theta_i \leq \tau + \Theta_i \), and the delay of the packet is less than \( \Theta_i \). If the replication process is currently replicating the packets of session \( i \), then the new packet can not be served immediately. The packet is placed in the corresponding session queue, and it will be copied the next time the session is served. Thus, the packet will be processed by time \( \tau + \Theta_i + f_i \), since the session will be added to the sessions queue at the latest by time \( \tau + f_i \).

The above Lemma proves that any packet will be served within a finite delay, that depends only on the number of multicast sessions in the system, and thus, the replication server is always stable. Note, that in the above proofs, we made absolutely no assumption about the arrival rate of packets for this or other multicast sessions, and thus the stability of the server is guaranteed even if the multicast traffic will lead some interfaces to congestion and instability. This is a much stronger result than simply demonstrating server stability if all interface queues are stable.

We can also notice that as long as the server can service a session within a bounded delay, the algorithm leads to stability. This observation has two important consequences:

1) The replication server can process sessions with a rate lower than once every packet transmission time, and the system will remain stable.

2) We can create servers that visit eligible sessions with different frequency, and offer varying delays at the replication server, without affecting the stability of the system.

**IV. Properties of Replication Servers**

We can extend the service discipline of the replication server to a set of disciplines, that will maintain system stability, and at the same time offer configurable latencies to different sessions. As we show in the previous Section, when a simple round-robin discipline is used by the replication server, the latency of any session is a function of the fan-out of all other sessions and the fan-out of the session itself. This might be unfair for some sessions with a low fan-out, since they might have to wait behind sessions with large fan-outs. In this Section we will show how we can modify the replication server to alleviate these problems.

First of all, we can notice that we can model the replication server as a general queueing server that offers service to a set of queues, similarly to to per-flow packet scheduler. The difference is that the service offered by the replication server is in terms of number of copies per unit of time, and the time needed to complete a replication is equal to the fan-out of the connection. This corresponds to the service offered by a general packet server in terms of bits per unit of time, where the time needed to serve a packet is determined by the size of the packet. We can see that there is an one-to-one correspondence between number of complete replications and packets, as well as fan-out and packet size. The capacity of the replication server \( RC \) is defined as the number of copies per time, where a copy is the function of attaching a session queue to a single interface. A complete replication of a session requires as many copies as the fan-out of the session. We can now use the ideas developed on packet scheduling to design replication servers with different properties.

We assume that a multicast packet is served by the replication server when it has been replicated to all interfaces. Let us denote with \( S_i(t_1, t_2) \) the service offered to multicast packets of session \( i \) by the replication process in the interval \( (t_1, t_2) \). The service offered is equal to the number of copies for this session during this interval. Note again, that this is not the number of packets of this session that have been replicated. These copies can correspond to one or more complete replications. Let \( f_i \) denote the fan-out of session \( i \) and let \( W_i^2(t_1, t_2) \) denote the actual number of packets that have been replicated during the interval \( (t_1, t_2) \) from session \( i \) to interface \( j \). Let also \( A_i(t_1, t_2) \) denote the number of packets arriving in the interval \( (t_1, t_2) \) for session \( i \).

We assume a general replication server, where a session is guaranteed a rate of copies \( \rho_i \). In other words, \( \rho_i \) determines how many times per second is a session copied.
to any interface. Note, that the rate of complete replications is equal to $f_i$, since a complete replication requires $f_i$ copies. We assume a set of servers that have the following property, referred as worst-case packet fairness within the context of packet schedulers [?, [?]:

**Definition 2:** A server is worst-case fair, iff for any interval of time $(t_1, t_2]$ during which session $i$ is continuously backlogged, it can guarantee service $S_i(t_1, t_2)$, that is bounded as

$$S_i(t_1, t_2) \geq \max((t_2 - t_1)\rho_i - \Theta_i, 0)$$

where $\rho_i$ is the rate allocated to session $i$, and $\Theta_i$ is a constant for that session. The sum of the rates allocated to all sessions must be less than the capacity of the server, or $\sum_{i=1}^{K} \rho_i \leq RC$.

### A. Delay Bounds

We can now prove the following Lemma:

**Lemma 3:** Any worst-case fair replication server maintains system stability and provides a maximum delay of $L_i = \frac{L_i + \Theta_i}{\rho_i} + f_i$ to any multicast packet, irrespective of the arrival traffic.

**Proof:** Consider any backlogged period $(t_1, t_2]$, and the time instants $\tau_1 < \tau_2 < \ldots < \tau_2$ where a replication for session $i$ is completed. Consider also the time instants $\hat{\tau}_1, \hat{\tau}_2, \ldots$ where the replication of some packets for session $i$ starts. Note, that for every $i$, we can write

$$\tau_i = \hat{\tau}_i + f_i$$

(2)

We can prove the Lemma for all time intervals $(\tau_i, \tau_{i+1}]$. For completeness we set $\tau_0 = t_1$ and $\hat{\tau}_0 = t_1$. Based on the definition of the worst-case fair replication servers, for any time $t$ in the interval $(\tau_i, \tau_{i+1}]$, we can write

$$S_i(\tau_i, t) \geq \max(0, (t - \tau_i)\rho_i - \Theta_i)$$

(3)

We are looking for the time instant $\tau_{i+1}$ that $S_i(\tau_i, t)$ becomes at least equal to $f_i$. At this time, a replication for the session is completed, and this means that all packets that have a arrived for this session in the interval $(\hat{\tau}_i, \tau_{i+1}]$ have been served. Note, that once the replication starts at time $\hat{\tau}_i$, any new arrivals after this time, will be processed during the next replication interval. By replacing in Eq. (3), we can see that $\tau_{i+1} \leq \tau_i + \frac{\Theta_i + L_i}{\rho_i}$. This means that the maximum delay $D$ of any packet that arrived at time $\hat{\tau}_i \leq t < \tau_{i+1}$ will be:

$$D \leq \tau_{i+1} - t$$

(4)

$$\leq \tau_i + \frac{\Theta_i + f_i}{\rho_i} - t$$

(5)

As we mentioned earlier, there is an one-to-one correspondence between replication servers and general packet schedulers. If any Worst-Case Fair Scheduler is used as a replication server, where packet arrivals happen when the first multicast packet of a session arrives or when a replication is completed, and packet sizes are equal to the fan-out of the session, we can maintain the above bound. Thus, any of the schedulers proposed in several publications as worst-case fair will offer the above properties [?], [?].

### B. Burstiness Bounds

One can realize, that due to the batching of packet arrivals at an interface, the burstiness properties of multicast sessions might be affected. In this Section we derive deterministic bounds on the burstiness increase of the sessions, as this is seen by the interface queues. Obviously, we can derive a burstiness bound only if we assume that arrival traffic has some burstiness bound. Thus, for these bounds, we will need to characterize the arrival process. We use a general burstiness constraint as defined in [?].

**Definition 3:** Given a non-decreasing function $b(t)$, we say that the arrival traffic is burstiness constraint by $b(t)$ if for any interval of time $(t_1, t_2]$,

$$A(t_1, t_2) \leq b(t_2 - t_1)$$

Note, that for a simple leaky bucket constraint traffic, $b(t_2 - t_1) \leq \sigma + r_i(t_2 - t_1)$, where $r_i$ is the rate of session $i$. We emphasize that the rate of session $i$ is totally irrelevant to the rate of replications allocated to that session.

We can prove the following Lemma:

**Lemma 4:** If a session arrival traffic is burstiness constrained, the output of the replication server is also burstiness constrained as:

$$W_i(t_1, t_2) \leq b(t_2 - t_1 + L_i + f_i)$$

For a leaky-bucket constrained arrival traffic, the output process is bounded by:

$$W_i(t_1, t_2) \leq \sigma + r_i(t_2 - t_1) + r_i(L_i + f_i)$$

**Proof:** Let us assume any time interval $(t_1, t_2]$. First we must take care of some boundary conditions. If the session is never backlogged during this time interval, and thus no service or no arrivals occur, the proof is trivial. Thus, we assume that some service was offered during this interval,
and the time instants that a replication is completed are \( \tau_0 \leq \tau_1 < \tau_2 < \ldots < \tau_M \leq t_2 \). As before, the time instants \( \hat{\tau}_1, \hat{\tau}_2, \ldots \hat{\tau}_M \) indicate the time instants where the corresponding replication of session \( i \) starts. We will assume that the session is backlogged at time \( t_1 \). If it is not, then there exists time \( t_1 > t_1 \), that the session first became backlogged. We can use the same proof starting at time \( t_1 \), and extending the proof to \( t_1 \) is straightforward since there are no arrivals or service in the interval \((t_1, t_1')\). We define \( \tau_0 \) as the maximum between the first time instant before time \( t_1 \) that a replication for this session was completed, and the time that the backlogged period that \( t_1 \) belongs into started. Also, \( \hat{\tau}_0 = \tau_0 - f_i \). The amount of traffic enqueued to any interface is equal to the sum of the data that have been copied to that interface during every replication time. From the definition of the replication server, the amount of data that are copied in the interval \((t_1, t_2] \) is less than or equal to all the packets that arrived in the interval \((\tau_0, t_2] \). Let us denote with \( \Delta T = t_2 - \hat{\tau}_0 \) the duration of this interval. We can write:

\[
\Delta T = (t_2 - t_1) + (t_1 - \hat{\tau}_0)
\]

We will distinguish two cases: If \( \hat{\tau}_1 \leq t_1 \), or in other words, the beginning of the period we are accounting for is during a replication for session \( i \), then

\[
\Delta T = (t_2 - t_1) + (t_1 - \hat{\tau}_1) + (\hat{\tau}_1 - \hat{\tau}_0).
\]

But from Lemma ?? we can conclude that \( (\hat{\tau}_1 - \hat{\tau}_0) \leq L_i \) and since \( t_1 \) is within the time that a replication is performed for session \( i \), \( t_1 - \hat{\tau}_1 \leq f_i \). Therefore,

\[
\Delta T \leq (t_2 - t_1) + L_i + P_i.
\]  

(8)

If \( \hat{\tau}_1 > t_1 \), then

\[
\Delta T = (t_2 - t_1) + (t_1 - \hat{\tau}_1) + (\hat{\tau}_1 - \hat{\tau}_0).
\]

But, the arrival traffic is constrained and thus

\[
W_i(t_1, t_2) \leq b(t_2 - t_1 + L_i + f_i).
\]  

(13)

Note, that for a leaky bucket constrained traffic, the above equation becomes:

\[
W_i(t_1, t_2) \leq \sigma + r_i(t_2 - t_1) + r_i(L_i + f_i)
\]  

(14)

C. Schedulability Analysis

As we show in the previous sections by using a worst-case fair replication server, we offer different latencies to multicast sessions, irrespective of the arrival traffic. The question we are trying to answer in this Section is based on the fan-outs of multicast sessions what are the range of latencies we can offer by adjusting the rates of service offered to each session. In order to make this decision we have several requirements to meet.

First of all, the replication server must be stable in terms of the amount of times that it visits each session. Obviously this translates to the requirement that:

\[
\sum_{i=1}^{K} \rho_i \leq RC.
\]  

(15)

Note, that the rate of session \( i \) in the replication server does not depend on the arrival rate of session \( i \), but strictly on the latency requirements. Let us assume that session \( i \) has a latency requirement of \( D_i \). Then for each session \( i \), the maximum latency offered by the server must be less than the delay requirement, or

\[
\frac{\Theta_i + f_i}{\rho_i} + f_i \leq D_i
\]  

(16)

It is also clear that in order to guarantee latency \( D_i \), we must have \( D_i > f_i \). Then, the rate that must be allocated to the session is:

\[
\rho_i \geq \frac{\Theta_i + f_i}{D_i - f_i}
\]  

(17)

By summing over all \( i \),

\[
\sum_{i=1}^{K} \rho_i \geq \sum_{i=1}^{K} \frac{\Theta_i + f_i}{D_i - f_i},
\]  

(18)

and from the capacity constraint (Eq. ??)

\[
\sum_{i=1}^{N} \frac{\Theta_i + f_i}{D_i - f_i} \leq RC
\]  

(19)

In other words, given the parameters \( \Theta_i \) of the server, the fan-out of the connections \( f_i \) and the delay requirements \( D_i \), by solving the equations ?? and ?? we can always determine if a set of rates \( \rho_i \) exists that will satisfy all the delay constraints.

V. Simulation Study

In this Section we present some simulation results of the simple round robin replication server as an indication of the average performance of the system. The worst-case
latency bounds, although tight, can be far from the latencies offered by the system, since not all multicast sessions are active at all times.

For comparison reasons we simulate three different types of systems. A maximum speed-up server based on the RAR scheme that uses a speed-up of \( N \) and is expected to provide the best performance since packets are replicated to interfaces immediately on arrival. A system using staging buffers, where multicast packets are placed in a single queue and one copy is performed during every packet time. And finally, the batch multicast system with per-session staging queues.

We assume that the system supports 64 interfaces, where the capacity of each interface is equal to \( \frac{1}{64} \) of the incoming peak rate capacity. For simplicity we assume equal size packets, and time is normalized in terms of the time needed to receive a single packet. Our arrival traffic model assumes an ON-OFF source with Pareto ON times and exponential OFF times to reflect a rather bursty traffic model. Arriving traffic is distributed uniformly across all multicast sessions. A total of 64 sessions is simulated in the system.

In all the plots Infinite Speedup corresponds to the RAR scheme with speed-up of \( N \). With “FIFO staging” we denote the scheme where a FIFO staging buffer is used, and the staging server can be copied to only one output at a time as described in Section ?? . In Figure ?? we show the average delays of one of the interfaces versus the fan-out of the multicast sessions for all different schemes. Note, that because traffic is uniformly distributed across all multicast sessions and interfaces, and all interface queues are stable, the average delays of all interfaces are similar. It is thus sufficient to concentrate on the observed delays of one of the interfaces. The average load of all interfaces is set to 80\% and the average ON time is just two packets. As we can see, the delays achieved by batch multicast are almost identical to those of the infinite speed-up scheme. The FIFO staging scheme results in slightly higher delays, although the system remains stable at all times. Remember, that as we discussed in Section ?? , as long as the average number of copies is less than the capacity of the system, the FIFO staging scheme can provide stable performance.

However, an implication of the FIFO staging scheme is that a large percentage of the delay is in the staging queue itself. Figure ?? shows the average number of total packets in the staging server for the FIFO staging scheme and the batch multicast scheme. As it can be easily seen, the batch multicast algorithm manages to maintain very small staging queues, and the quality-of-service for the multicast traffic will mainly depend on the scheduling mechanisms and load of individual interfaces. Contrary, in the FIFO staging scheme queueing is dominated by the staging buffer. Figure ?? shows the interface queue size for the tagged interface as a function of the fan-out, and for the same simulation. We can observe, that even for this sim-
ple scenario, where the burstiness of multicast sessions is low, most of the queueing in FIFO staging is performed in the staging queue and the interface queues remain small. Contrary, in the batch multicast and RAR schemes queue sizes at the interfaces are similar, and queueing at the interface dominates the delays. This means that in FIFO staging we have no means of controlling the delays and load of individual interfaces, whereas in the batch multicast scheme performance can be controlled by the interface queueing discipline. This becomes especially important when we mix unicast and multicast traffic.

In Figure ?? we show the delay versus load curve for different loads, when the multicast fan-out is set to 32. We set the average ON time to 20 packets. As it can be seen, irrespective of load, the batch multicast scheme achieves average delays comparable to the infinite speed-up scheme. The FIFO staging scheme offers slightly higher delays.

The scenario where the importance of the batch staging scheme becomes even more prominent is when some of the interfaces face temporary instability or they are overloaded. In this case, the traffic management components of the system must maintain good multicast performance for un-congested interfaces, and forward only a percentage of the multicast traffic to the congested interfaces. Since the system is mainly targeted to edge router applications, if one user is congested, other users must not be prevented from receiving the content they choose to, if they are well under their interface capacity. In Figure ?? we show the delay versus load curve in the case where some of the interfaces in the system are congested. We assume that 128 multicast sessions are active in the system. Out of these sessions, 64 distribute traffic uniformly across the interfaces, and the rest 64 transmit traffic only towards 16 specific interfaces. This means that at least 16 of the interfaces are under constant overload. We plot the average delay of one of the un-congested interfaces. As it can be easily seen, the batch multicast scheme offers delays and bandwidth almost identical to the infinite speed-up scheme. Contrary, the FIFO staging scheme leads the system to instability even for loads under 50%.

VI. IMPLEMENTATION DISCUSSION

The batch multicast scheme is based on a per-multicast session queueing, and one can argue that the state information makes the approach expensive. However, this is not the case for multicast implementations. All IP multicast protocols require state information for each multicast group, that identifies the outgoing interfaces [?]. In the case of ATM or MPLS multicasting, state is maintained on a per VC or Label-Switched Path in order to determine the outgoing interfaces. The additional state required by the batch multicast algorithm is two pointers. We claim that this state information does not impose a significant overhead.

Note also, that unicast traffic can be also forwarded using the same approach. One has to simply introduce an additional $N$ queues, one queue for each interface, where packets destined to interface $i$ are staged in the corresponding queue. Multicast and unicast queueing become uniform in this case, and the implementation complexity is further reduced.

In several systems where the access technology of choice is ATM, such as DSL and PON systems, one VC is allocated to each end user. In addition to other functions, these systems require support of VC merging and/or segmentation and re-assembly of IP packets depending on the trunk technology. Both of these functions require already the incoming session queues in order to re-assemble complete packets, before they are multiplexed on the egress
VC. In these cases, support of the multicast batching algorithm becomes even simpler, since all the data structures are available to begin with.

VII. CONCLUSIONS

In this paper we presented an efficient mechanism for replicating multicast packets with high fan-out. The mechanism is based on per-session staging queues, where packets are replicated in batches to outgoing interfaces. We showed that the replication server is stable, irrespective of arrival patterns, and the scheme can operate with no speed-up. This is the first algorithm reported in the literature that allows high fan-out multicasting without speed-up. Farther, we extended our model and showed how a range of scheduling algorithms can be adapted for the replication server, and can adjust the latencies of different multicast sessions.

It would be interesting to systematically study the statistical performance of the system. Indeed, one can model the system as a single message polling system with constant service times. Future work can include this analysis, as well as buffer management techniques for multicast traffic.

Note also, that the availability of such architectures in the network can enable simpler instantiations of layered multicasting techniques when combined with a a diff-serv type of model [9]. In layered multicasting, a layered source coding algorithm is used. Different layers are distributed over different multicast sessions, and end systems can choose the number of layers that they can receive based on their congestion. Such a method, although efficient from a content distribution perspective, introduces additional state in the network, since multiple multicast groups are needed for forwarding a single multicast stream. Alternatively, one can mark packets of different layers using a diff-serv type approach, and the multicast enabled routers can use simple traffic management techniques to forward a subset of the traffic to different interfaces. In order for routers to support such a mechanism however, they must deploy deterministic replication mechanisms such as the burst multicast described in this paper.

REFERENCES