On the Use of On-Demand Layer Addition (ODL) with Multi-Layer Multicast Transmission Techniques

Vincent Roca
LIP6 - CNRS, theme Réseaux et Performances, Paris, FRANCE
INRIA Rhône-Alpes, projet Planete, Grenoble, FRANCE
vincent.roca@inrialpes.fr - http://www.inrialpes.fr/planete/people/roca/

ABSTRACT

This work deals with the multicast transmission of data using multiple multicast groups. One reason to use multiple groups is to address the potential heterogeneity of receivers. But there is a risk that some of the groups are not used and sending data to a group with no receiver has many costs that are often underestimated. As IP multicast becomes widely deployed and used, these issues may well compromise its scalability.

In this paper we introduce a new protocol, ODL (On Demand Layer Addition), which *enables a source to use only the layers that are actually required by the current set of receivers.* We describe its behavior when used with several kinds of packet scheduling schemes (cumulative or not) and different scenarios (one-to-many versus many-to-many). We have implemented the ODL protocol, integrated it in the MCL multicast library and we report several experiments that assess its benefits.

1. INTRODUCTION

One trend of research in multicast communications is to rely on several multicast groups. This is a scalable and efficient way of sending information to a set of highly heterogeneous receivers, e.g. in terms of processing power, of transmission capabilities, or of power range.

This is used by video applications [5][24]. To accommodate the potential receiver heterogeneity, the source uses a layered data coding and transmits each layer in a separate multicast group. Users join as many groups as made possible by their available bandwidth. If the associated congestion control mechanism indicates that a user should reduce its incoming traffic, then he leaves one or more groups. As soon as all the users behind the bottleneck router have left, this branch of the multicast distribution tree is pruned, thereby reducing the load on the router [15][23]. Scalability is guaranteed by the fact that adding or dropping a layer is a local receiver decision that does not require any source implication (except for some minor synchronization purposes [23]).

NGC2000 - Second International Workshop on Networked Group Communications; Stanford University, Palo Alto, California, USA, 8-10 November 2000.

ALC (Asynchronous Layered Coding) [13] is a recent scalable reliable multicast protocol proposed by the RMT IETF working group that relies on multiple multicast groups. Unlike the case of layered video streams, each ALC receiver must get all the data. ALC can be used by many different applications for either "on-demand", "streaming" or "push" delivery of data. In "on-demand" delivery, data is sent continuously and receivers arrive at any time. This mode is well suited to the distribution of popular files (e.g. a video-clip, the latest Linux distribution, etc.). In "streaming mode" a receiver typically remains joined for a long period, receiving many objects (e.g. images) produced in real-time. ALC builds upon many previous works in the area of multi-layer data organization schemes [22][18], congestion control for multi-layer transmission schemes [15][23], and forward error correction (FEC) [17][14].

By lack of an appropriate mechanism, an application relying on multi-layer transmission will define a fixed number of multicast groups, usually given as an argument at start up. There is a risk that some of these groups are not used by any receiver, especially with continuous file transmissions (e.g. the number of receivers may largely vary between peak and off-peak periods). There is also a risk that "badly educated" people create sources with a highly overestimated number of layers.

Minimizing the number of multicast groups defined by a source is all the more important as these applications are to become popular. In this paper we introduce a new protocol, ODL (On Demand Layer addition), that is well suited to layered transmission schemes like ALC. It relies on the presence of a *receiver-oriented* congestion control scheme for layered transmissions (e.g. RLM [15], RLC [22]) where each receiver chooses to add or drop a layer within the fixed set of available layers. ODL's goal is to *enable the source to use only the layers that are actually required by the current set of receivers.*

The rest of the paper is organized as follows: we introduce the motivations in section 2. Section 3 details the ODL protocol under many different circumstances. Section 4 discusses the initialization of the various protocol timers. Then section 5 introduces its implementation, and presents several experimental results. Section 6 discusses related work. Finally we conclude.

2. MOTIVATIONS FOR USING ODL

2.1 How Many Layers?

The question: *"how many layers should a source use"*

turns out to be rather complex as the group membership is by default not known by the source. Because of the lack of an appropriate tool, we believe that content providers will define a (largely overestimated) fixed number of layers, without bothering whether they will be used or not. On the opposite, with ODL a source can start with a small number of layers and progressively add or drop a layer according to the receiver requirements. Because of various limitations (e.g. CPU, network access, codec, etc.) the source will usually define an upper bound to the number of layers it can offer.

2.2 Various Costs associated to a Multicast Group

The packets sent to a group with no receiver are usually dropped by the first-hop multicast router. But there are additional costs that are often underestimated: (i) the *state kept by multicast routers* and (ii) the *useless traffic*.

Forwarding State Kept by Multicast Routers:

In order to assess the *memory cost* of a group in each multicast router, we analyzed the `mrouted` (version 3.8) distribution. We found that a group leads to the allocation of (more than) 100 bytes of state information, no matter whether there are receivers beneath this router or not (see next section). This example is of course highly protocol and implementation dependent (e.g. [21] discusses techniques to reduce this cost). Yet it shows that state can be a source of scalability problems (e.g. for 1,000 groups, at least 100 kilobytes of costly vendor specific memory are necessary).

The Case of Dense-Mode Protocols:

Dense-mode routing protocols, the "old legacy generation of protocols", rely on the periodical flooding of data. This is the case for DVMRP and PIM-DM. This feature greatly increases the network traffic and the router load even on branches with no receiver.

A second consequence is that *all the routers keep information for this group*, even if they do not belong to the multicast distribution tree [1]. For instance, a router running PIM-DM keeps a context with the source and group addresses, the state and a timer for each group in "prune state".

With this class of protocols ODL avoids both packet transmissions and forwarding state for unused multicast groups.

The Case of Sparse-Mode Protocols:

Because of their limitations, dense-mode protocols are progressively replaced by sparse-mode protocols that rely on an "explicit join" mechanism. The most popular is PIM-SM. This protocol first creates a unidirectional (i.e. source dependent) shared (i.e. by all receivers) tree, rooted at a well known RP (Rendez-vous Point). In a second step, each leaf router contacts the source to establish a per-source shortest path tree. If in theory switching from a shared tree to a per-source tree occurs above some session bandwidth threshold, in practice it occurs immediately, at the first packet reception (most vendors have set the threshold to zero kbps) [8].

ODL has little interest in case of a PIM-SM domain in "shortest path tree" mode since: (i) forwarding state is only kept along the distribution tree and (ii) without any receiver, packets are dropped by the first hop router. ODL gains are only local to the source, the LAN and the first hop router (fewer packets sent and processed). This is different in "unidirectional shared tree" mode, as group knowledge is kept by the RP. In case of an empty group (i.e. when ODL is of interest) packets are anyway sent to the RP (which may be located far from the source), thereby creating both state in the RP/first hop router and useless traffic.

The Case of MSDP for Domain Interconnection:

The evolution of inter-domain multicast led to the development of PIM-SM/MBGP/MSDP [1]. The role of MSDP (Multicast Source Discovery Protocol) is to inform remote domains of the presence of a source in the local domain. Whenever a new source becomes active in a given domain, its local MSDP peer announces its presence (`SA` message) to all directly connected MSDP peers, who then forward the message to other peers. This information is periodically refreshed and is also cached to reduce subscription latency [1]. This periodic flooding to the (intentionally limited number of) MSDP peers, and the information state they keep takes place even if the group has no member. Using ODL has in that case a clear advantage.

The Case of Future Multicast Routing Protocols:

Because of its limitations, the PIM-SM/MBGP/MSDP architecture is only a medium-term solution. Single source multicast (e.g. Express [12]) is a good candidate for the future. It relies on a (root, address) tuple, makes the tree source rooted, and can be implemented using (slightly modified versions of) PIM-SM/IGMPv3 [3]. The previous discussion on using ODL with PIM-SM/per-source tree applies here also.

The Case of Reflectors and Host-Based Multicast:

But multicast connectivity is fragile and not as widely deployed as expected. There are many reasons for that, including a lack of interest from providers, a lack of proper management tools, important scaling problems, etc. [8]. Using reflectors [7] as an immediate solution, and host-based multicast [6] as a longer-term solution offers the possibility to connect unicast and multicast domains.

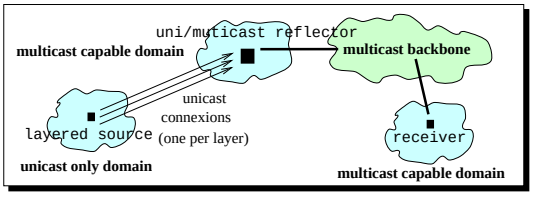


Figure 1: Layered source using a remote unicast/multicast reflector.

Let us consider a source located in a unicast domain (figure 1). This source reaches the multicast backbone through a remote reflector, located in a multicast capable domain. Each layer is sent in a separate unicast connection to the reflector. Using ODL enables the source to stop transmissions to the reflector for unused layers. The situation is exactly the same with host-based multicast.

3. THE ODL PROTOCOL

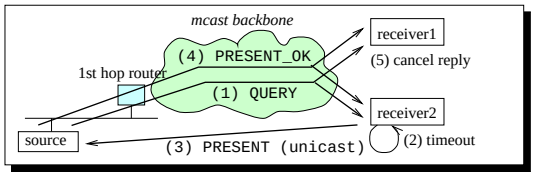
We assume (i) that each layer is carried on a *separate multicast group*. Therefore *the source* adds or drops a layer

by sending or not packets to a particular multicast address. Once the source stops sending new packets to a group, the soft-state kept by multicast routers (section 2.2) for this group slowly disappears (e.g. `mrouted` uses a default 5 minute timer). *A receiver* adds or drops a layer by joining or leaving the associated multicast group. We also assume (ii) that transmissions are *cumulative*. Section 3.6 analyzes the use of ODL when these assumptions are not met.

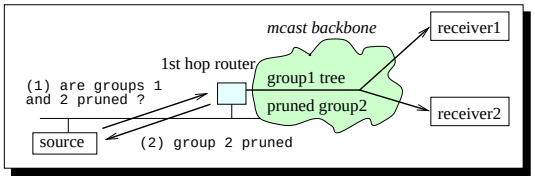
3.1 Sketch of the protocol

ODL has two working modes:

- by default, ODL behaves as a *pure end-to-end protocol*, working in a request / response mode. Here ODL has no requirement, neither on the router functionalities nor on the multicast routing protocol in use.
- when appropriate, ODL can also *poll the first-hop multicast router*. Even in that case, end-to-end requests / responses are still required during certain stages. Here some assumptions are made on the first-hop router and the multicast routing protocol in use (section 3.7).



(a) end-to-end mode



(b) router polling mode

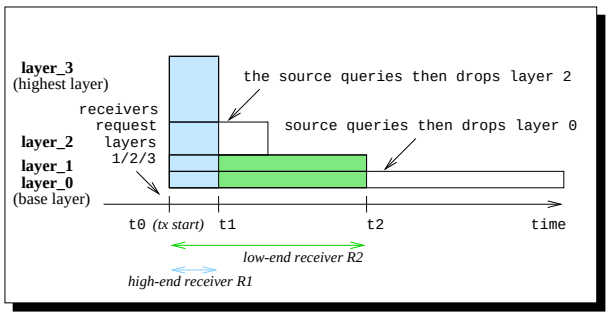
Figure 2: The two working modes of ODL.

ODL relies on both the source and the receiver sides: it is the *receiver's responsibility* to ask for additional layers when appropriate (e.g. if its congestion control module requests it) and to respond to `QUERY` messages. It is *up to the source* to create additional layers when requested by a receiver. Of course the source can refuse (e.g. if its maximum number of layers has been reached).

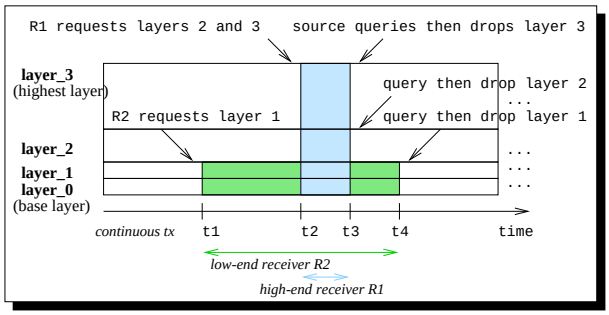
In pure end-to-end mode, the source checks periodically that each multicast group is used by at least one receiver by sending `QUERY` messages on each target layer (figure 2 (a)). If no receiver responds, this (useless) layer is dropped. *In router polling mode*, the source polls the first-hop router that already keeps some information on group membership (figure 2 (b)). In particular this router can know if the tree is pruned or not.

ODL is designed so that the *feedback is always unicast'ed to the source*. A feedback suppression mechanism is added in order to avoid source implosion, but unlike protocols like IGMP, SRM [9], etc. where answers are multicast'ed, ODL does not require many-to-many multicast.

Synchronous versus Asynchronous Start:



(a) Synchronous start



(b) Asynchronous start

Figure 3: Transmissions on each layer with a synchronous or asynchronous start of the receivers; with ODL only the grey areas are used.

Figures 3 (a) and (b) compare layer management at the source with and without ODL. The high-end receiver, R_1, joins all four layers, and the low-end receiver, R_2, only joins the first two layers. We consider two kinds of applications:

- an application doing a *one time file transfer* (figure 3 (a) where an MMG data organization [4] is assumed): all the receivers must be ready before the transmission starts (synchronous start). This is similar to the ALC "push mode".
- an application doing *continuous file transfers* (figure 3 (b)): receivers can arrive at any time (asynchronous start). This is similar to the ALC "on-demand mode".

Using ODL enables the source to remain as close as possible to the actual requirements of the receiver set (in light grey for R1, and light/dark grey for R2). For instance layers 2 and 3 are quickly dropped after that R_1 has finished receiving the whole file. Without ODL the layers remain active (i.e. packets are sent) pointlessly (i.e. no receiver) also during the periods corresponding to the white areas. Note that so far we did not assume any latency between the time

Table 1: ODL messages.

type	sent by	used by	kind of transmission	description
INFO_REQ	receiver	source	unicast to source	a receiver asks for information
INFO	source	receivers	multicast on layer 0	information on all active layers
LAYER_REQ	receiver	source	unicast to source	a receiver asks for a new layer
ADD_LAYER	source	receivers	multicast on layer 0	a new layer has been added
DROP_LAYER	source	receivers	multicast on layer 0	a layer is dropped
QUERY	source	receiver	mcast on target layer	does anybody receive this layer?
PRESENT	receiver	source	unicast to source	yes, this receiver uses the layer
PRESENT_OK	source	receivers	mcast on target layer	PRESENT acknowledgment

the last receiver leaves and the time it is detected by the source (see section 4).

3.2 Detailed Description for the Sending Side in End-to-End Mode

We first consider the sending side and we assume (i) that there is only one source and (ii) that receivers are not on the same host as the source. In particular it means that the source is not a receiver (these cases are addressed later on in this paper).

At session start, the source uses a single layer. We assume that the address of the associated group has been communicated by some external means.

To know if there is at least one receiver for a given layer, the source periodically transmits a `QUERY` message on the target group. Because transmissions are not reliable, the `QUERY` message is sent several times (optimistically only 2 times in the current implementation)[1]. If no answer (`PRESENT` message) is received after a given time, the layer is dropped. Therefore the source issues a final `DROP_LAYER` message (to let everybody know this layer no longer exists) and avoids sending any packet anymore. The associated multicast tree slowly disappears as the routers soft-state times out. On the contrary, if the source receives a `PRESENT` message, the layer is kept and an acknowledgment (`PRESENT_OK` message) is echoed to the group. In order to detect duplicates, `QUERY` messages include an identifier that is written back in `PRESENT` and `PRESENT_OK` messages.

ODL uses two timers at the sending side: (i) the *soft-state_timer* (periodicity of the `QUERY` messages) and (ii) the *drop_timer* (waiting time before dropping a layer once the `QUERY` has been sent). This is discussed in section 5.

Layer Dependency:

Using a cumulative scheduling scheme means that layer i can only be dropped if there is no layer $> i$. In practice conflicts often take place. In figure 4 we assume that the last receiver leaves at time t_0. For each layer this departure is detected at the next query stage. Because the layers timers are not synchronized, layers 0 and 1 must wait that layer 2 has been dropped before being dropped in their turn.

[1]If all the QUERY are lost by all the receivers, the layer is wrongfully dropped. Interested receivers will ask for it later on, e.g. when receiving the following DROP_LAYER or a future INFO.

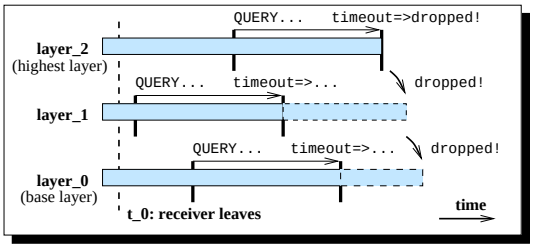


Figure 4: Interdependence of layers with a cumulative scheme; lower layers drop is delayed by layer 2.

An Optimization; The Suppression of Signaling on Lower Layers:

A consequence of layer dependency is the possibility to stop the polling on lower layers. Only the highest layer is polled, thereby reducing the total ODL signaling overhead. If the highest layer is dropped, the source issues `QUERY` messages on the remaining layers, drops useless layers (if any) and only polls the new highest layer.

3.3 Detailed Description for the Receiving Side

A new receiver first joins layer 0, waits a little bit for information ((`INFO` message) and sends to the source an `INFO_REQ` message if nothing is received. The answer (`INFO`) informs each receiver of the current situation: number and features of the layers in use. The receiver can then join one or more layers depending on what his congestion control module says.

A receiver who wants to benefit from an additional layer not yet available sends a `LAYER_REQ` message to the source using unicast. The source replies with an `ADD_LAYER` message if he agrees, with an `INFO` message (specifying that the maximum number of layers has been reached) otherwise. Note that this mechanism *only adds a small latency during the layer addition process*, in the order of one RTT (`LAYER_REQ/ADD_LAYER` exchange), and the time required to create the multicast tree is almost the same, with or without ODL[2].

Another major task of a receiver is to answer to `QUERY` messages (figure 2 (a)). For the sake of scalability, the `PRESENT` message is not returned immediately but after a random time. At timeout and if no other message has been received, the receiver sends a `PRESENT` message in unicast to the source and waits for an acknowledgment. If no

[2]In fact layer addition latency is essentially delayed by the congestion control module that dictates when to add a layer.

PRESENT_OK message is received after a given time, then a new PRESENT message is sent. After a given number of unsuccessful tries (e.g. if the connectivity with the source is lost) the receiver drops the layer.

3.4 The Case of Multiple Sources

The case of many-to-many transmissions is more complex. In particular the sources can be heterogeneous and can offer a different number of layers. Therefore *ODL must enable an individual treatment of each source.* To do so, the sender's private address (address/port number) of a message is always communicated to the receiver. If the multicast library (section 5.1) always reads ODL messages using the `recvfrom` or `recvmsg` socket system calls, then the sender's address is known and the receiver can reply in unicast. This feature is used for instance when a receiver requests a layer to a particular source: the LAYER_REQ is directed to this source by specifying its private address and using unicast.

3.5 The Case of Receivers on the Same Host as a Source

A receiver on the same host as a source can usually receive data at full transmission rate and will therefore request all the layers. This is a problem if this host is the only one to use the upper layers as it would defeat ODL. The same situation occurs if the source is also a receiver. To avoid it, the source uses a TTL of 0 for layers where all the receivers are local. Therefore packets never leave the host. But whenever there is at least one foreign receiver, the TTL value specified at application start is used instead. The algorithm is given below:

```
on receiving a LAYER_REQ message for a new layer:
    if (the message comes from the local host)
        add layer in ''local transmission mode''
        (i.e. with a TTL of 0);
        send ADD_LAYER (TTL of 0);
    else
        do the normal processing (ADD_LAYER...);
    activate the layer;

on receiving a PRESENT message:
    if (the message comes from the local host)
        note that there is at least one local receiver;
        do nothing else yet...
    else
        do the normal processing
        (PRESENT_OK, reset the drop timer);

on drop_timeout:
    if (there is a local receiver)
        go into ''local transmission mode''
        (i.e. with a TTL of 0);
    else
        drop the layer;
```

Note that this behavior can be generalized to hosts located at some distance (i.e. within a given TTL scope) from the source (e.g. on the same LAN).

3.6 The Case of Non Cumulative Packet Scheduling Schemes

ODL can be used with non cumulative approaches too. For instance, with Destination Set Grouping (DSG) [2], data is sent over separate multicast groups at different rates and a receiver chooses *the* appropriate group. To receive data faster, a receiver changes (instead of adding) of group. If the new group is not already active, then he first asks for it with a LAYER_REQ. Because it generates some delay, the receiver keeps on listening to the previous group until the new one is available. There are two differences yet: (i) the ODL signaling previously sent only on the base layer is now *sent to all the groups*, and (ii) dropping or adding a group is immediate as there is no layer dependency.

3.7 Description of ODL in Router-Polling Mode

With a source-rooted tree the first-hop multicast router can usually say if the tree is pruned or not. Therefore the source can directly poll this router rather than receivers, using an extension (to be defined) to IGMP. This extension is called "reverse-IGMP" (revIGMP) as it works in the opposite way (with IGMP the local multicast router collects information on group membership). The advantages are:

- less traffic (e.g. hardly any QUERY/PRESENT/PRESENT_OK message)
- no risk of source implosion
- takes advantage of information already available

The details of revIGMP, its applicability and limitations are out of the scope of this paper.

4. ANALYTICAL ANALYSIS OF THE PROTOCOL TIMERS

4.1 The DROP Timer

The `drop_timer`, maximal waiting time before removing a layer, depends on:

- the RTT to the farthest receiver: max_RTT (request / response exchange) and
- the maximum time a receiver can wait before answering to QUERY: $T_{maxwait}$.

$$T_{drop} = max\_RTT + T_{maxwait} \quad (1)$$

In practice max_RTT is rather difficult to evaluate [11] [9]. To avoid this extra complexity[3], we define a "reasonable" constant: $reas\_max\_RTT$ (1 s in the current implementation) and an adaptive parameter: $RF \geq 1$, the Robustness Factor. RF is controlled by a multiplicative increase/additional decrease (MIAD) algorithm given below and we use:

$$max\_RTT = RF * reas\_max\_RTT$$

[3]Note that we do not try to find an accurate maximum RTT estimation (unlike TCP where a weighted average is used). Having a rough large upper bound is sufficient in our case and using a MIAD scheme limits the possible flip-flop behavior of ODL.

```
the source issues a DROP_LAYER with a new identifier;
if (the source receives no LAYER_REQ with the same
    identifier within two max_RTT)
        // the drop_timer value was probably correct,
        // so decrease RF...
        RF = max(RF - add_factor, MINIMUM_RF_VALUE);
        max_RTT = RF * reas_max_RTT;
if (the source receives a LAYER_REQ with the same
    identifier)
        // the drop_timer value was too short,
        // so increase RF...
        RF = min(RF * mult_factor, MAXIMUM_RF_VALUE);
        max_RTT = RF * reas_max_RTT;
```

4.2 The SOFTSTATE Timer

The `softstate_timer` is the period between two `QUERY` requests. This timer can be managed more or less aggressively.

Aggressive Polling:

An aggressive management scheme tries to minimize the delay between the departure of the last group member and the layer drop. Because transmissions are faster on higher layers than on lower ones, it is natural to check the presence of receivers more frequently on higher layers, i.e. to use a smaller value for the `softstate_timer`: $T_{ss}(k)$. We can define a cost function for layer k: $Av\_Cost(k)$, as the average number of packets sent between the time the last receiver leaves and the time ODL detects that this layer is no longer used. Therefore, if we assume a uniform distribution:

$$Av\_Cost(k) = \frac{(T_{ss}(k) + T_{drop})}{2} * rate(k)$$

where $rate(k)$ is the transmission rate for this layer. We want to keep $Av\_Cost(k)$ below a fixed threshold for all layers: Av_Cost. Thus:

$$T_{ss}(k) = \frac{2 * Av\_Cost}{rate(k)} - T_{drop}$$

We don't want $T_{ss}(k)$ to become negative (which means that the target cost is not feasible for this layer), so we define a minimum value: min_T_{ss}. Likewise we require that $T_{ss}(k)$ remains below a maximum time, max_T_{ss} . It ensures that an unused multicast group can not last longer than max_T_{ss} seconds. Thus:

$$T_{ss}(k) = min(max\_T_{ss}; max(min\_T_{ss}; \frac{2 * Av\_Cost}{rate(k)} - T_{drop})) \quad (2)$$

Fixed Rate Polling:

A simpler solution is to use a constant sampling rate in the $[min\_T_{ss}; max\_T_{ss}]$ interval for all layers, thereby ignoring their individual features.

4.3 Scalability Management

The ODL feedback suppression mechanism has limitations in case of very large groups. Several parameters, analyzed in section 5.5, can influence the average number of `PRESENT` messages returned. In order to avoid any risk of source implosion, ODL adapts its `QUERY` sending rate. This is easily done by making the lower bound of the `softstate_timer` depend on the number of replies obtained: $N_{replies}(k)$ (this is in practice a weighted average). Let $max_{replies}$ be the maximum number of replies desired per second. Thus we must ensure that:

$$\frac{N_{replies}(k)}{T_{ss}(k)} \leq max_{replies}$$

and the new lower bound of equation (2) becomes:

$$max(min\_T_{ss}; \frac{N_{replies}(k)}{max_{replies}}) \quad (3)$$

Reducing the polling frequency does not contradict ODL's goals. It only happens with large multicast groups for which the probability of having all the receivers leave simultaneously is low in "on-demand" mode.

Another complementary solution is to advertise a new $T_{maxwait}(k, t+1)$ in each `QUERY` message and to make it depend on the previous $N_{replies}(k, t)$. If $N_{replies}(k, t)$ is above $max_{replies}$, then $T_{maxwait}(k, t+1)$ is increased, thereby reducing the number of replies (figure 8 (b)), otherwise it is reduced. A MIAD algorithm can be used for that.

5. PARTIAL EVALUATION

5.1 The ODL Building Block

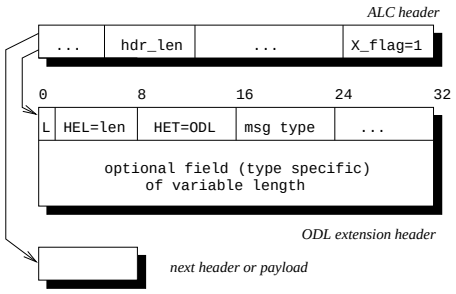


Figure 5: ODL extension header format appended to the ALC header.

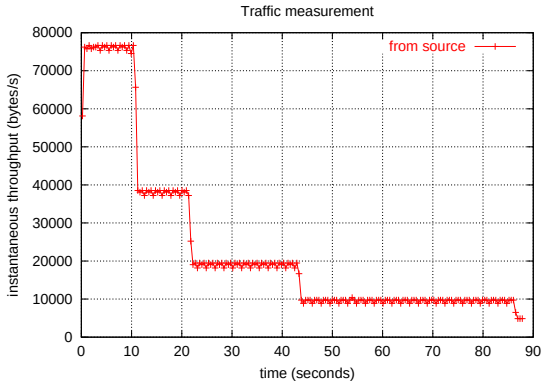
We have implemented the ODL protocol as a building block (~1000 lines of C) that we integrated in our application-level MCL library [19]. This library relies on UDP/multicast IPv4. The ODL messages are piggy-backed to data packets thanks to the concept of *extension header* (figure 5) of ALC [13]. Any number of ODL headers can be attached as long as the MTU is not exceeded. This solution greatly reduces the transmission cost for downward signaling (source to receivers).

5.2 Test setup

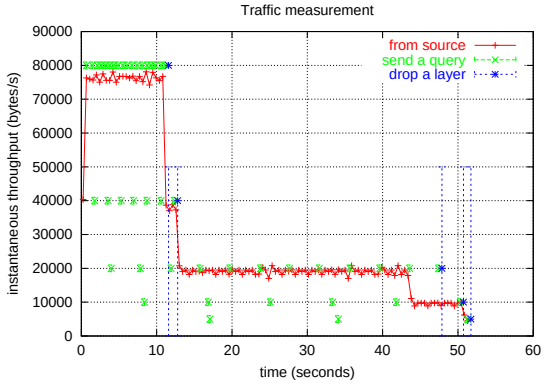
The tests include a source that performs a one-time file transfer (400 kbytes), uses five layers, a cumulative scheduling scheme, and a synchronous start. The situation is similar to that of Figures 3 (a). The average cumulative throughput is respectively 5, 10, 20, 40 and 80 kbytes/s. There are two receivers: the high-end receiver, R_1, subscribes to all five layers (0 to 4), and the low-end receiver, R_2, only subscribes to the first three layers (0 to 2). Three hosts

connected to an Ethernet LAN are used, one for the source and the `tcpdump` tool, the two others for the receivers. The optimization mentioned in section 3.2 is not enabled.

5.3 Benefits of ODL with Aggressive Polling



(a) without ODL



(b) with ODL

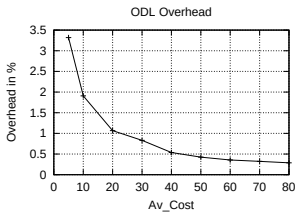
Figure 6: Traffic at the source without/with ODL.

Figures 6 (a) and (b) show the resulting traffic at the source. The parameters are: $Av\_Cost = 40$ packets, $min\_T_{ss} = 0.2$ s, $max\_T_{ss} = 20$ s, $T_{maxwait} = 0.4$ s and $RF = 1$. In figure 6 (b), the five vertical lines after time 11 s show the drop of each layer. The isolated points illustrate the sending of a `QUERY` message on a layer.

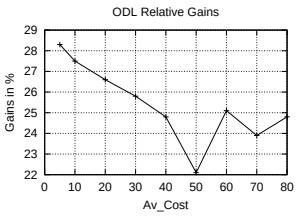
We see that ODL periodic checks are efficient: the higher the level, the faster the receiver departure is detected and transmissions stopped. Layer 3 is dropped only a few seconds after that R_1 has left (at time 10.9 s). An analysis of `tcpdump` traces show that only 64 data packets have been sent during this interval on layer 3, which remains in line with the $[0; 2 * Av\_Cost]$ target. For layers 1 and 0, 65 and 74 useless packets are respectively sent after the departure of R_2 (at time 43.5 s). Once again it remains inferior to the $2 * Av\_Cost$ maximum.

From table 2 we see that *ODL reduced by 25.1% the total number of bytes on the LAN and by 22.5% the total number of packets.* Of course this result greatly depends on the scenario in use: number and features of the set of receivers. At the same time *ODL's overhead* (i.e. the number of bytes introduced for ODL signaling) *remains very low* as it only represents 0.54% of the total traffic (in bytes). This is largely due to the fact that ODL messages sent by the source are often piggy-backed into data packets. Using ODL only on the highest layer (section 3.2) would further reduce ODL's overhead.

5.4 Impacts of the Av_Cost Parameter



(a) Overhead



(b) Gains

Figure 7: ODL's overhead versus gains as a function of the Av_Cost parameter.

We carried out the same experiments while varying the Av_Cost parameter which controls the query frequency. The overhead introduced by ODL and its benefits are shown in figures 7(a) and (b). As the Av_Cost parameter increases, the overhead goes asymptotically to zero (fewer queries). This overhead is anyway very small, below 1% of the total amount of data sent as soon as $Av\_Cost > 20$ packets.

The "gains" curve is more complex. As expected, in the $[5; 50]$ interval the smaller the Av_Cost parameter, the faster the detection of unused layers. This is no longer true above 60. The reason is the following: queries are sent every max_T_{ss} seconds on layer 0 when $Av\_Cost \geq 60$ packets. It turns out that a query is sent on layer 0 *immediately after* the departure of receiver R_2. Therefore the source is immediately informed (instead of waiting an additional max_T_{ss} period) and drops layer 0. (in fact layer dependencies (section 3.2) may delay a bit the effective drop of layer 0)

5.5 Scalability of the Feedback Suppression Mechanism

We have simulated the scalability of ODL's feedback suppression mechanism. The major parameters are:

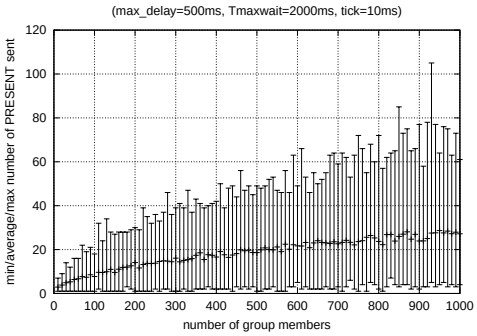
- *the maximum waiting time*, $T_{maxwait}$, before answering to a `QUERY`. Each receiver is assigned a waiting time uniformly distributed in the $[0; T_{maxwait}]$ interval.
- *the maximum unidirectional delay*, max_delay: For each receiver we chose a delay uniformly distributed in the $[0; max\_delay]$ interval. We assume that paths are symmetric (i.e. $RTT = 2{*}delay$)[4] and use $max\_delay = 500ms$ (1 second RTT) to consider the (rather unfavorable) possibility of very far nodes.

[4]The presence of asymmetric paths is not meaningful for this kind of simulation

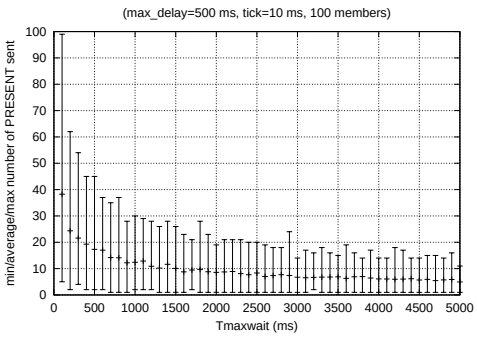
Table 2: Detailed measures without and with ODL, $Av\_Cost = 40$ packets.

	without ODL	with ODL
Total traffic from source (data+ODL, including UDP/IP hdr)	2,176,710 bytes or 3,951 pkts	1,631,425 bytes, or 3,061 pkts
Total traffic to source (ODL replies, including UDP/IP hdr)	N/A	2,720 bytes, or 48 pkts
Total ODL overhead	N/A	8,776 bytes, or 0.54%
Gains made possible by ODL	N/A	25.1% (bytes), or 22.5% (pkts)

- *the internal time tick interval:*. Time is discrete within the MCL library. Therefore the timer controlling the sending of a `PRESENT` message has a limited granularity. The higher the timer precision, the lower the number of `PRESENT` messages sent simultaneously.



(a) as a function of the number of group members



(b) as a function of the Tmaxwait parameter

Figure 8: Min/average/max number of PRESENT messages sent.

Figure 8 (a) illustrates this scalability for a 2 second maximum waiting time (ratio $T_{maxwait}/max\_delay = 2000/500 = 4$). For each x-axis value, 1000 different random topologies are created. With 100 members, there are between 1 and 18 `PRESENT` messages with an average of 7.7, which remains acceptable. More important, the curve shape does not show any exponential increase but stabilizes. Figure 8 (b) studies the effect of $T_{maxwait}$ for a fixed number of 100 members: the higher the $T_{maxwait}/max\_delay$ ratio, the better the efficiency of feedback suppression.

5.6 Discussion of the results

- *the smaller the Av_Cost parameter, the higher the chances of a fast detection of unused layers.* But it remains *probabilistic* and using longer query periods also increases the chances of unpredictable results. There is a *strong analogy with the signal sampling theory.* Unfortunately in our case the maximum "signal" frequency and thus the optimal sampling rate is not known!
- Anyway the *gains made possible by ODL are significant*, even if they greatly depend on the configuration. On a LAN, with aggressive polling, we observed an average 25% reduction in the number of bytes sent. With a WAN configuration the benefits mentioned in section 2 also apply.
- Simulations have shown that basic feedback suppression is efficient. Section 4.3 suggests two additional improvements.

6. RELATED WORK

Dynamic source-oriented adaptation:

Dynamic source adaptation has often been proposed. For instance RTCP (RTP Control Protocol) [20] returns feedback information to the source(s). Its goals are to "monitor the quality of service" and to "convey information about the participants in an on-going session". ODL only addresses one particular issue not covered by RTCP, so both protocols bring separate benefits.

[24] describes the SAMM (Source Adaptive Multi-layered Multicast) algorithm. The source adjusts dynamically the number (like ODL) and the individual rate of each layer of a layered video stream thanks to feedback information sent by the receivers and/or routers. Unlike ODL, SAMM mixes both the adaptation and congestion control functionalities, it requires that routers implement some form of priority-based packet discarding and perform feedback merging.

Membership size estimation:

Several mechanisms have been proposed to estimate the size of a multicast session. [5] introduces a probabilistic polling method through several polling rounds, each with a higher reply probability. [16] describes a single polling round technique. [10] adds refinements to these works. The EXPRESS single-source multicast routing protocol [12] advocates the use of a router support for counting. Yet one possible implementation of this service [3] does not mention this possibility. ODL does not care about the (estimated) number

of members and only returns a boolean answer: "yes or no there is at least one receiver" which is much simpler.

In fact ODL can also be associated with a protocol giving an estimation of the number of receivers (e.g. if it is required by another building block). As long as this estimation is above a given threshold, ODL is useless. As soon as it falls below this threshold, ODL is enabled for a rapid and accurate behavior when the last receiver leaves.

7. CONCLUSIONS

In this paper we have: (i) discussed the issue of *idle multicast groups* (i.e. when there is no receiver currently interested in), and (ii) introduced *a new protocol: ODL* that enables a source to be informed in real-time of the presence and possibilities of receivers. ODL follows either an end-to-end or a direct router polling approach and is *very beneficial to multi-layer multicast transmission schemes*, be they cumulative (e.g. ALC) or not (e.g. DSG). It avoids the use of idle groups whose cost is often under-estimated: state kept in multicast routers, periodic tree management traffic (flood/prune of dense-mode protocols, `SA` messages of MSDP, etc.), useless traffic with unicast/multicast reflectors. This is of much importance to guaranty the *scalability of IP multicast* as it becomes widely deployed and used.

We have implemented ODL (available on the author's home page), integrated it within our MCL multicast library [19] and made several experiments to assess its benefits.

8. ACKNOWLEDGMENTS

The author thanks L. Vicisano for his comments and the idea of "reverse-IGMP" and the anonymous reviewers.

9. REFERENCES

[1] K. Almeroth. The evolution of multicast: from the mbone to inter-domain multicast to internet2 deployment. *IEEE Network, Special Issue on Multicasting*, January 2000.

[2] M. Ammar and L. Wu. Improving the performance of point to multi-point arq protocols through destination set splitting. In *IEEE INFOCOM'92*, May 1992.

[3] S. Bhattacharyya and C. Diot. *Deployment of PIM-SO at Sprint*, March 2000. Work in Progress <draft-bhattach-diot-pimso-00.txt>.

[4] S. Bhattacharyya, J. Kurose, D. Towsley, and R. Nagarajan. Efficient multicast flow control using multiple multicast groups. In *IEEE INFOCOM'98*, February 1998.

[5] J. Bolot and T. Turletti. Scalable feedback control for multicast video distribution in the internet. In *ACM SIGCOMM'94*, September 1994.

[6] Y-H Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, June 2000.

[7] T. Cicic and H. Bryhni. *Multicast-unicast reflector*, January 0000. unpublished document, available at URL: http://www.ifi.uio.no/ tarikc.

[8] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *IEEE Network*, pages 78–88, January 2000.

[9] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *IEEE SIGCOMM'95*, 1995.

[10] T. Friedman and D. Towsley. Multicast session size membership estimation. In *IEEE INFOCOM'99*, March 1999.

[11] M. Handley, B. Whetten, R. Kermode, S. Floyd, L. Vicisano, and M. Luby. *The Reliable Multicast Design Space for Bulk Data Transfer*, March 2000. Work in Progress, <draft-ietf-rmt-design-space-01.txt>.

[12] H. Holbrook and D.R. Cheriton. Ip multicast channels: Express support for large-scale single-source applications. In *ACM SIGCOMM'99*, September 1999.

[13] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff. *Asynchronous Layered Coding (ALC): a massively scalable reliable multicast protocol*, July 2000. Work in Progress: <draft-ietf-rmt-pi-alc-01.txt>.

[14] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff. *Reliable multicast transport building block: Forward Error Correction codes*, July 2000. Work in Progress: <draft-ietf-rmt-bb-fec-01.txt>.

[15] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM'96*, October 1996.

[16] J. Nonnenmacher and E. Biersack. Optimal multicast feedback. In *IEEE INFOCOM'98*, February 1998.

[17] L. Rizzo and L. Vicisano. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), April 1997.

[18] L. Rizzo and L. Vicisano. Reliable multicast data distribution protocol based on software fec techniques. In *Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communcation Systems (HPCS'97), Greece*, June 1997.

[19] V. Roca. *A library for heterogeneous multicast distributions: concepts, architecture and use*, January 2000. Work in Progress, http://www.inrialpes.fr/planete/people/roca/.

[20] J. Rosenberg and H. Schulzrinne. *Sampling of the Group Membership in RTP*, February 2000. Request For Comments 2762.

[21] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. In *IEEE INFOCOM'00*, March 2000.

[22] L. Vicisano. Notes on a cumulative layered organisation of data packets across multiple streams with different rates. Research Note Note RN/98/25, University College London (UCL), May 1998.

[23] L. Vicisano, L. Rizzo, and J. Crowcroft. Tcp-like congestion control for layered multicast data transfer. In *IEEE INFOCOM'98*, February 1998.

[24] B. Vickers, C. Albuquerque, and T. Suda. Adaptive multicast of multi-layered video: Rate-based and credit-based approaches. In *IEEE INFOCOM'98*, February 1998.