A Fragmentation Scheme for Multimedia Traffic in Active Networks *

Sheng-Yih Wang and Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
E-mail: {swang,bb}@cs.purdue.edu

Abstract

Multimedia data are usually very large in size. Fragmentation of multimedia data unit is inevitable when they are transmitted through networks. Active networks are becoming popular and the active technologies are being applied to various interesting problems. When applying active technologies on multimedia data, however, the problem of fragmenting large packets still exists. Furthermore, new issues emerge when active capsules are fragmented.

In this paper, we propose a new fragmentation scheme which addresses the unique need of active networks and utilizes the special properties of active networks. We propose an algorithm to fragment the data at the transport layer which can minimize the overhead. Preliminary experimental result shows that the scheme works well under realistic scenario, with less than 5% of overhead.

1 Introduction

Multimedia objects such as audio, video and image are usually very large in size. For example, one MPEG-1 movie in NTSC video quality (which we digitized from a VHS video tape) has an average size of 8617 bytes/frame. The average I-frame size is even larger (17997 bytes/frame). When transporting multimedia data in the networks, the sizes of the payload are usually larger than the Maximum Transmission Unit (MTU) of the underling physical networks. For example, the MTU of Ethernet, the most popular link-layer technology, is 1500 bytes. Therefore it is inevitable that the multimedia data have to be fragmented into smaller-size units when transmitting through physical networks. In the past, this issue is not particularly significant because the application programs can simply assume that the lower level protocols (such as IP and ATM) can reliably deliver reasonably large packet (64 KB in IP and unlimited size in ATM AAL5) and let the lower level protocols handle the fragmentation process.

In recent years, the researches on active networks [7] show some potentials in providing better network services to the emerging applications which are rich in content and require large bandwidth. For example, video gateways, which used to be implemented in application-level, may be possible to be implemented using active networks [6]. However, since any network traffic, no matter it is from active network or not, has to be transported using conventional link-layer technologies such as Ethernet or ATM, the problem of fragmenting large packets still exists in active networks. Furthermore, some new issues emerge when active capsules are fragmented into smaller units. Therefore new fragmentation schemes are needed for the active communication environment.

In this paper, we propose a new fragmentation scheme which addresses the unique need of active networks and utilizes the special properties of active networks to achieve the task. The paper is organized as follow: In section 2, some alternatives of the fragmentation schemes are introduced and the issues specific to active networks are discussed. In section 3, the proposed scheme is discussed and an example is given to illustrate the ideas. Two new socket system calls are introduced to help the application programs take advantages of the new network services. In section 4 we present some preliminary experimental results of our scheme. Finally, in section 5 we discuss some unresolved issues and outline our future works.

2 Fragmenting Large Data Units

There are several alternatives to handle the fragmentation problem. They are:

- **Fragment at network layer.** This is the current practice when passive protocols such as TCP or UDP send large packets. The internals of the payload are not revealed to the network and the fragmentation points are decided arbitrarily and

*This research is partly supported by a grant from NSF under NCR-9405931

up to the network layer protocol. IP fragmentation and ATM Adaptation Layer (AAL) are examples of this approach. The advantage of this approach is that the fragmentation process is transparent to the higher layer protocols (both transport and application layers).

- **Fragment at transport layer.** Transport layer can also fragment data into units of appropriate size before transmission. However, current practice of IP networks allow very large packet (64 KB) to be sent and let the IP layer handle the fragmentation.
- **Fragment at application layer.** The application layer is the best place to decide how to fragment the data because it has complete information about the data to be sent. The idea of *application-level framing* [2] and *integrated layer processing* [1] explore some aspects of this alternative. However, applications may not be able to know the limitation of the underlining networks. Therefore it may not be able to make the best decision in case there are several alternatives available. Practically speaking, there is no mechanism in current network architectures to support the cooperation between the application programs and network protocol in dealing with data fragmentation.

Although active networks are promising network architectures for the next-generation network, some issues specific to the active networks exist regarding to the fragmentation of data. They are:

- Fragments of the same capsule may travel different routes. The intermediate active routers may never have the chance to execute the active programs inside a capsule simply because the successful execution of the active programs needs all the fragments of the same capsule. This is not a problem in conventional IP networks because the reassembly only occurs at the receiver, not at the routers.
- Reassembly cost of active capsules in the intermediate active routers may be too high. If an active capsule can be processed only after all the fragments are received by the active router and the active capsule is very large (which results in a lot of fragments), then the time of waiting for all fragments may be too long for time-sensitive data and the overhead to reassembly all the fragments for the active routers and re-fragment them after processing may be too much.
- If the active capsules are fragmented arbitrarily, the fragments may contain partial logical data units which will increase the complexity of the active programs inside the capsules because the active programs have to be aware of the possibility of partial data due to fragmentation and have to handle the problem by themselves.

Although researchers of active networks recognize the importance of the fragmentation problem [4], the issues of fragmentation mechanisms in active networks have not been addressed in current research yet. The fragmentation scheme described in this paper represent our first attempt to address this important problem.

3 Proposed Scheme

3.1 Design Goals

Our design goals of a fragmentation scheme for active networks are:

- Fragmentation points should correspond to the natural boundaries inside the data which can be meaningfully managed, if it is possible to do so.
- Fragmentation policy should not divide a data into fragments unless it is necessary.
- For the data that can not be fragmented at the natural boundaries, the fragmentation/reassembly costs should be minimized.

3.2 Basic Ideas

The basic idea of our proposed scheme is: first, instead of letting the network layer blindly fragment the application data, the application gives out *hints* to the lower layer protocols on how the data can be fragmented. The application also tells the lower layer protocols how it will handle the data for both fragmented and un-fragmented cases. In addition, the network layer can pass the MTU information up to the application, so the application can adjust its behavior to fit the network constraints when possible. We believe that the ACTive Protocol at transport layer (ACTP) is a logical choice to combine and process both information from application layer and network layer.

The advantages of handling the fragmentation process inside the active protocol are:

- The fragmentation process can be done in an uniform, application-independent way.
- By giving out hints only, the applications don't need to handle the complexity of actually fragmenting the data.

- The active programs don't need to handle the complexity of dealing with partial data.

3.3 Type of Hints

There are two type of hints that can be passed to the active protocol: fixed-size hint and variable-size hint. Fixed-size hint contains a single value which represents the required fragment size for the data. It is useful when the nature of the data requires that it can be divided into fixed-size fragments. For example, sound samples in audio-conferencing are usually fixed-size data.

Variable-size hint contains an array of breakpoints that can be applied to the data. Note that fixed-size hint is actually a special case of variable-size hint. Since we envision that there are many cases where fixed-size hint will be useful and the programming interface of fixed-size hint can be greatly simplified, we separate it out as another type of hint.

3.4 An algorithm for fragmentation at active protocol

According to our design, the actual fragmentation process occurs inside the active transport protocol. The fragmentation process utilizes the informations from both the applications (hints) and network infrastructure (MTU). There are many possible ways to fragment the application data using the provided information. In this section we provide an algorithm which can minimize the number of fragments generated based on the provided information. Figure 1 present the algorithm in C-style pseudo-codes. A proof of optimality is in appendix A. The algorithm below is designed based the following constraints:

- The algorithm tried to fill as many data sub-units (determined by the breakpoints inside the data unit) as possible into a fragment whose size is upper bounded by the maximum fragment size.
- Partial data sub-units will not be placed in the same fragment as whole data sub-units.
- The algorithm will fragment the data at non-breakpoints only when it is impossible to fit the data into the maximum fragment size by using breakpoints only.
- The algorithm will not re-order the data within a data unit.

Note that in the pseudo-codes the function call `Fragment()` take two parameters (fragmentation point, fragment size).

```
//Input parameters:
//   breakpoints[]  — array of breakpoints in ascending order
//   data_size      — data unit size
//   bc             — number of breakpoints
//   mss            — maximum fragment size
base = 0;
for (j = 0; j < bc; j++) {
   if ((breakpoints[j] - base) == mss) {
      Fragment(breakpoints[j], breakpoints[j]-base);
      base = breakpoints[j];
   }
   else if ((breakpoints[j] - base) > mss) {
      if ((j > 0) && (base != breakpoints[j-1])) {
         Fragment(breakpoints[j-1], breakpoints[j-1]-base);
         base = breakpoints[j-1];
         }
         if ((breakpoints[j] - base) > mss) {
            while ((breakpoints[j] - base) > mss) {
               base += mss;
               Fragment(base, mss);
            }
            if (breakpoints[j] != base) { /* still something left */
               Fragment(breakpoints[j], breakpoints[j]-base);
               base = breakpoints[j];
            }
         }
      }
}
if (base != data_size) { /* some more leftovers */
   Fragment(data_size, data_size-base);
}
```

Figure 1: Active Fragment Algorithm

3.5 Programming Interfaces

We propose the following new system calls to the socket API to provide the services described above. They are:

- `int send_actp(int s, void *msg, int len, int hints[], void *seg_prog, void *unseg_msg, unsigned int flags)` — `send_actp()` is similar to the `send()` system call in usual socket library. It is augmented with three more arguments which pass the hints to the active protocol. The argument `hints` is an array of breakpoints. If `hints` is for fixed-size hint, a single negative integer value will be provided whose absolute value is then used by the active protocol as the proposed fragmentation size. The argument `seg_prog` and `unseg_msg` correspond to the active programs which handle the fragmented and un-fragmented case.

- `int getsockopt(int s, IPPROTO_ACTP, ACTP_MTU, void *optval, int *optlen)` — A new socket option at the active protocol layer (or *level* in terms of socket library convention) called ACTP_MTU is introduced. When the application calls `getsockopt()` to retrieve MTU information, active protocol can perform a path-MTU discovery to retrieve this information and cache it for future use.

3.6 Example

To illustrate the fragmentation scheme we proposed, consider the case when one frame of a MPEG video is to be sent via ACTP/IP/Ethernet network. The size of the packet is 5300 bytes, which is larger than the maximum fragment size of 1286 bytes (1286 bytes is the MTU of Ethernet minus necessary spaces for IPv6 header, ANEP header [3] and active programs). The application pass th list of hints { 8, 1251, 2433, 4096, 5300 } to ACTP and ACTP obtains the constraint of maximum fragment size (1286 bytes) from the network layer. The list of hints correspond to the offsets from the beginning of the frame where a new slices of macroblocks starts. According to the algorithms, the first fragmentation point is 1251 because the maximum possible sub-units from the beginning of the data that can fit into an 1286-bytes fragment is at 1251 bytes. The next fragmentation point is 2433 because the fragment will be too large if ACTP fragment at the the next breakpoint. The next fragmentation point is 3719, which dose not correspond to any breakpoints in the hint list. The reason is that because data sub-unit between offset 2433 and 4096 is too large for a 1286-bytes fragments, ACTP has no choice but to fragment at the middle of the data sub-units. The next fragmentation point will be 4096, which finish the data left behind before the breakpoint 4096. The last fragmentation point will be 5300, which is the last fragment of this frame.

4 Experiment

To determine the overhead incurred in our fragmentation scheme, we run the algorithm on an entertainment quality MPEG-I video. The purpose of this experiment is to get a rough idea on how many extra bytes are needed using our scheme under realistic scenario.

The MPEG video clip we used as input was digitized from the video tape of the movie *Jurassic Park* to the resolution of 320x240 at 30 frames/sec. The video clip consists of 2904 frames, which has approximately 97 seconds of running time. The average picture size is 8617 bytes, and the average size of I, P and B frames are 17997, 15473, and 4866 bytes respectively. We choose the maximum fragment size to be 1286, which is the MTU of Ethernet minus the spaces for IPv6 Header, ANEP header and an active program of 100 bytes. It is apparent that almost all the video frames are larger then the maximum fragment size and need to be fragmented. In our experiment, we pass a hint list consist of the offsets from the beginning of a video frame which correspond to the starting points of the slices inside the video frame. The assumption is that since slices are the largest meaningful sub-units inside a video frame, the application and the active programs know how to deal with slices directly, therefore it makes sense to fragment the video frames at the boundary of slices to reduce the complexity of active programs. By fragmenting at slice boundaries, the active program can now focus on dealing with complete and meaningful data sub-units. Therefore the processing of the fragments may be optimized or may even be done independently.

We compare the overhead in our scheme to the case where the data are simply fragmented to the size of MTU minus header size. The result shows that the average overhead is 4.7% of the total data size, and the maximum overhead for a single frame is 12.8%. The worst case occurs when the video frame consists of a lot of slices which are slightly larger than the maximum fragment size.

From this preliminary experimental result we feel that the proposed scheme is performing well under the realistic experiment scenario with low overheads.

5 Future Work and Concluding Remarks

Although preliminary experimental and theoretic results show that the proposed scheme is promising, there are some issues that need to be addressed further:

- The overhead due to the fragmentation process is high (more than 10%) in some cases. Although the average overhead is less than 5%, it may be desirable to further reduce this overhead. One possible approach is to loosen the constraints of the orderly delivery within a data unit. By rearranging the breakpoints, it is possible to produce less fragments, thus reduces overhead.

- Sometimes handling partial data unit in active programs may not be a critical issue at all. This is because the additional complexity an active program incurred by handling partial data unit strongly depends on the nature of the data. In these cases the best way to handle the data is simply to fragment it into pieces whose sizes are the maximum fragment size. Any scheme proposed to handle fragmentation problem should leave this possibility as an option to the application.

- In our scheme, we assume that the size of active programs is small comparing to the data they process. Therefore we only consider the problem of fragmenting the data portion of the active capsules, not the active program portion. We believe that active programs should be small enough to be useful [8]. This view interpolate well with the view of most of the researchers in this field which states that "The primary function of the active network is communication and not computation." [5].

Although active networks are drawing a lot of attentions from the researchers, it is still not clear how active technologies can provide substantial advantages to persuade a change in the network infrastructure. We envision that a potential way of utilizing the active network technology is to provide customized application-level QoS at the network layer [8]. In fact, we are not the only people who hold this view. In [9], the authors proposed a network architecture motivated by application-level QoS. In the active network mailing list [4], the author further elaborates this point of view and states that "an active networking approach should have a significant edge over a traditional one in providing application-level QoS".

We believe that network-aware application in cooperation with application-aware network is the way for future network infrastructure to support emerging applications.

References

[1] Torsten Braun and Christophe Diot. Protocol implementation using integrated layer processing. In *Proceedings of the ACM SIGCOMM*, August 1995.

[2] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM SIGCOMM*, 1990.

[3] D. Scott Alexander et al. Active network encapsulation protocol (ANEP). Available at http://www.cis.upenn.edu/~switchware/ANEP/, February 1998.

[4] Joseph B. Evans and Gary J. Minden. Active networking archive. Web site at http://www.ittc.ukans.edu/Projects/ActiveNets, 1998.

[5] AN Working Group. Architectural framework for active networks. Available at http://www.cc.gatech.edu/projects/canes/arch/arch-draft.ps, July 1998.

[6] Shunge Li and Bharat Bhargava. Active Gateway: A Facility for Video Conferencing Traffic Control. In *Proceedings of COMPSAC'97, Washington, D.C.*, pages 308–311. IEEE, August 1997.

[7] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communication Magazine*, 35(1), January 1997.

[8] Sheng-Yih Wang. Approaches to multimedia traffic management and control. Technical report, Department of Computer Sciences, Purdue University, October 1997.

[9] Geoffrey G. Xie and Simon S. Lam. An efficient network architecture motivated by application-level QoS. Technical Report CS-97-Xg-2, Computer Science Department, Naval Postgraduate School, May 1997.

A Proof of optimality of the fragmentation algorithm

The optimality of the fragmentation algorithm can be proved as follows. First let's observe that the breakpoint before a data sub-unit whose size A is larger than the maximum fragment size M is always a fragmentation point. Since we require that no partial data sub-unit will be combined with whole data sub-unit, the next breakpoint is also a fragmentation point. In this case at least the $\lceil \frac{A}{M} \rceil$ fragments are needed and our algorithm produces exactly $\lceil \frac{A}{M} \rceil$ fragments, therefore the algorithm is optimal.

Following the above reasoning, the only case we need to consider is where all the data sub-units have sizes which are less than the maximum fragment size. Assume that the algorithm produces a solution A which produces a sequence of fragmentation points at a_1, a_2, ..., a_n. Let's suppose that the algorithm didn't produce the optimal result. That is, there are other ways of fragmentation which produce less fragments. Consider any such optimal solution S, which produces a sequence of fragmentation points at s_1, s_2, ..., s_m. It is apparent that m must be smaller than n because S is optimal. Now we assert that $s_i \leq a_i$ for all i from 1 to m. First, $s_1 < a_1$ because the algorithm requires that a_1 contains the maximum possible breakpoints that can be included. Second, $s_2 \leq a_2$ because (a_1, a_2) contains the maximum possible breakpoints that can be included, and (s_1, a_2) contains at least one more breakpoint than (a_1, a_2). Following similar arguments, we conclude that $s_i \leq a_i$ for all i from 1 to m. Now consider the last fragment of solution S. It start from offset s_m to the end of the data. Since $s_m \leq a_m$ and $m < n$, there is a $k \leq n$ such that $s_m < a_k$. Then from the requirement of the algorithm there are at least two fragments between s_m and the end of the data because it require at least two fragments between a_m and the end of the data. But this is a contradiction because by assumption s_m is the last fragmentation points. Therefore the optimality of the algorithm is proved.