Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 5, pp. 727-734

Towards Low Delay Sub-Stream Scheduling

W. Guofu, D. Qiang, W. Jiqing, B. Dongsong, D. Wenhua

Wu Guofu, Dou Qiang, Wu Jiqing,
Ban Dongsong,Wenhua Dou
National University of Defense Technology
School of Computer Science
Changsha, Hunan, P.R.China
E-mail: {gfwu,qdou,jqwu,dsban,whdou}@nudt.edu.cn

Abstract: Peer-to-Peer streaming is an effectual and promising way to distribute media content. In a mesh-based system, pull method is the conventional scheduling way. But pull method often suffers from long transmission delay. In this paper, we present a novel sub-stream-oriented low delay scheduling strategy under the push-pull hybrid framework. First the sub-stream scheduling problem is transformed into the matching problem of the weighted bipartite graph. Then we present a minimum delay, maximum matching algorithm. Not only the maximum matching is maintained, but also the transmission delay of each sub-stream is as low as possible. Simulation result shows that our method can greatly reduce the transmission delay.

Keywords: P2P streaming, scheduling, sub-stream, weighted bipartite graph, matching

1 Introduction

The emerging peer-to-peer (P2P) systems have appeared to be the most promising driving force for video streaming over the Internet. By distributing the workload to a large number of low-cost computing hosts such as PCs and workstations, one can eliminate the need for a costly centralized server and at the same time improve the system's scalability. There are already commercial products emerging, e.g., *PPLive* [1], *CoolStreaming* [3].

Any P2P streaming system consists of two distinct but related components: (i) an Overlay Con- struction mechanism, and (ii) a Content Scheduling mechanism. To improve the performance of P2P streaming systems, many studies focus on the overlay construction. However, the content scheduling mechanism also has greatly impaction on the performance. Carefully designed scheduling mechanism could have a better tradeoff among maximum streaming rate, minimum transmitting delay and control overhead. The overlay construction falls into two categories: tree(s) and mesh. In the mesh overlay, the pull method which is very similar to that of Bit-Torrent protocol [2]is widely used. Recent result [5]shows that the protocol of unstructured mesh overlay outperforms the traditional multi-tree approaches much in many aspects.

Several media content scheduling mechanisms for mesh-based system have been proposed. *CoolStreaming/DONet* [3] and *Chainsaw* [4] proposes pull-based scheduling framework. JM Li [6] designs a scheduling algorithm to manipulate the order of data blocks to improve the transmission efficiency in P2P streaming system. M Zhang [7] defines priorities for different blocks according to their rarity property and their emergency property, and tries to maximize the average priority sum of each node. Based on the traffic from each neighbor, a node in *GridMedia* [8] subscribes the pushing packets from its neighbors at the end of each time interval. *Pulsar* [9] combines push-based operations along a structured overlay with flexibility of pull operations. *LStreaming* [10] uses sub-streams in the push-pull streaming system.

Copyright © 2006-2010 by CCC Publications

This paper presents an effectual content scheduling strategy which can largely reduce the content transmission delay meanwhile remaining the main advantage of the pure pull method. First, the original stream is divided into k sub-streams, and each sub-stream has the same rate. A node requests one sub-stream instead of one content block from its neighbors. Then the sub-stream scheduling problem is transformed into the matching problem of weighted bipartite graph. The well known $Hungarian$ Algorithm which solves the maximum matching problem is ameliorated. Not only maximum matching is reserved by the new improved algorithm, but also the transmission delay of each sub-stream is as low as possible. The main difference between our scheme and *LStreaming* is the choice of available neighbors.

The following paper is organized as follows: Section 2 describes our motivation. In Section 3, we present our algorithm for the sub-stream scheduling problem. Simulation result shows In Section 4. Finally, we conclude our work in Section 5.

2 Motivation

2.1 Delay analysis of the pull-based scheduling strategy

The main drawback of pull-based scheduling strategy is that content blocks suffer from long delay. Now we give a quantitative analysis showing why this strategy resulting in long transmission delay. In such system, the media content is divided into equal size blocks, each of which has a unique sequence number. Every node (including the source node) periodically broadcasts all of its neighbors a bit vector called Buffer Map (BM) which represents the availability of useful blocks in its buffer pool. According to the announcement, each node decides from which neighbor to ask for which blocks, and periodically sends requests to its neighbors for the desired blocks. When a neighbor receives the request, it puts the desired blocks into its output queue, waiting to send out. For the efficiency of information exchange, BM and requests will only be sent periodically so that dozens of packets can be mapped into a single packet. We denote the interval between two buffer map packets or two request packets as T.

Figure 1 shows a typical process that a content block goes from one node to a neighbor. At the time t_1, a fresh block arrives in node A. Because the BM packets will only be sent at the beginning of each time slot, the useful information will wait until the time t_2, when a new time slot begins. After a period of time t_{d1}, at the time t_3, the BM packet reaches at node B. Time t_{d1} contains two components: propagation latency and transmit latency. The propagation latency lies on the length of links between node A and node B, while the transmit latency lies on the length of packet and the available bandwidth. The width of the shadow represents the transmit latency. Because the length of BM and request packets is small, we ignore the latency. At the time t_3, node B would make a design whether or not to request the fresh block from node A. As the same reason, node B sends the request packet at the time t_4. At the time t_5, node A receives the request packet, and then it puts the fresh block to the output queue. After a queue delay t_w, the fresh block is sent out. Because the length of content block is large, we can't ignore the transmit latency of the packet. At the time t_7, the fresh block is received by node B. Now we can compute the one hop delay, which is the interval between the time t_1 and t_7. Apparently, t_p and t_r are uniform random variable on the length T of time slot, and they are independent. Their average values are both $T/2$. We suppose the packet with largest waiting time will be sent first, the blocks before the time t_1 reached at node A at the same time slot will be sent before the fresh block, so the queue delay t_w is $T - t_p$. Thus, the average latency $t_{1-delay}$ for one packet in one hop can be computed as

$$t_{1-delay} = E[t_p + t_{d1} + t_r + t_{d2} + t_w + t_{d3}] = \frac{3T}{2} + 3\overline{t_d} + \frac{l}{u} \qquad (1)$$

Where $\overline{t_d}$ is the average end-to-end delay, and l is the length of content block, and u is the average upload bandwidth of nodes.

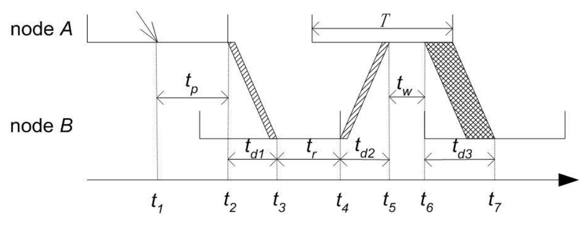
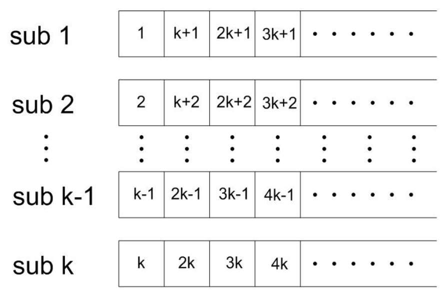


Figure 1: one hop latency

2.2 Using sub-stream to reduce the one hop latency $t_{1-delay}$

We can reduce the latency by adopting a shorter cycle T', but this will bring in more information packets. Meanwhile the three end-to-end delays can't be avoidable. A naive solution is that when node A receives the fresh content block, it directly sends it to node B. In this case, the one hop latency is only $\overline{t_d} + \frac{l}{u}$. But node B may receive repeated blocks from different neighbors under blind packets scheduling. An efficient way is that node A sends specifically sub-stream(s) to node B, while other neighbors send another sub-stream(s) to node B. Different sub-streams are not intersected. Node B decides to subscribe which sub-stream from which neighbor, and requests for sub-stream(s) instead of content block(s). Theoretically, node B requests for sub-stream(s) only once, and then just waits to receive media content. Figure 2 shows the creation



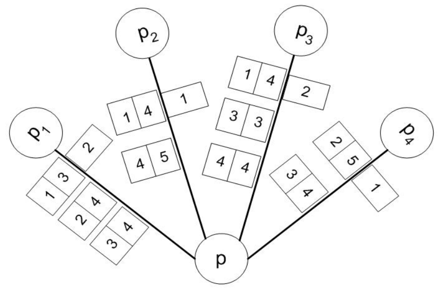


Figure 2: content blocks in different sub-streams Figure 3: the information peerp collects

of sub-streams. The original media content is divided into blocks with equal size, and each block has a unique sequence number, counting from number 1. Suppose we divide the original stream into k sub-streams, then what content blocks each sub-stream has is expressly described in figure 2. Which sub-stream a content block with sequence number m belonging to is determined by the following function $f(m)$.

$$f(m) = \begin{cases} m \mod k & \mathtt{if}\ x \mod k \neq 0 \\ k & \text{else} \end{cases} \tag{2}$$

3 Sub-stream Scheduling

In our framework, participating peers adopt gossip protocol to self-organize into unstructured mesh overlay. In this section, we study the problem of sub-stream scheduling, that is to say a node subscribe which sub-stream from which neighbor. We suppose in the initial period, the node uses pull-based method to get the content blocks. After that, push-based method works.

3.1 Exchanging information

Peers periodically broadcast sub-stream information and available upload bandwidth to neighbors. The period can be long, because other cases may trigger a node to broadcast the information or send the information to chosen neighbors. When a node joins the system, all of its neighbors send information packets to it actively. The information includes the sub-streams the neighbor can provide $\{s_{ij}\}$, latency of each sub-stream from the source to the neighbor $\{h_{ij}\}$ and the available upload bandwidth of the neighbor $\{c_i\}$. s_{ij} =1 presents $neighbor_i$ can provide $sub-stream_j$, else $s_{ij} = 0$. h_{ij} presents the number of hops of $sub-stream_j$ from the source to $neighbor_i$. If $s_{ij} = 0$ then $h_{ij} = +\infty$. c_i is not the real bandwidth, but it presents the multiples of one sub-stream rate. For example, $neighbor_i$'s available upload bandwidth can support transmitting two sub-streams to local node, then $c_i = 2$. After a short period, the local node gets information from all of its neighbors. Suppose the original stream is divided into 4 sub-streams. Figure 3 shows the information that $peer_p$ collects from all of its neighbors. The information on the left side of every connection presents the sub-streams neighbors can provide and their latency starting from the stream source node, while the number on the right side presents how many sub-streams the neighbor's available upload bandwidth can support. For example, in figure 3, neighbor p_1 can provide sub-stream 1, 2, 3 respectively, and the corresponding latency is 3, 4, 4 hops respectively. Meanwhile the available bandwidth between p_1 and p can support 2 sub-streams.

3.2 Constructing weighted bipartite graph

More than one neighbors can provide the same sub-stream, so we should make a decision which neighbor is more suitable as a provider for one sub-stream. Here we transform the sub-stream scheduling problem into the matching problem of weighted bipartite graph. First, we construct the weighted bipartite graph, according to the information collecting from neighbors. The weighted bipartite graph can be expressed by a quadruple group $G = (X, Y, E, W)$, where X presents the set of sub-stream providers, Y presents the set of sub-streams, E presents the set of connections between X and Y, and W presents set of weight on each connection. There is at most one connection between any two elements in X and Y respectively. So if a neighbor's upload bandwidth can support several sub-streams, we should characterize this situation in our bipartite graph. Here, we allow same elements to coexist in set X to deal with this situation. If $neighbor_i$'s available upload bandwidth can support c sub-streams, then we copy c same elements of $neighbor_i$ into set X. When all neighbors are put into set X, we complete the construction of set X. Next we build the connections between X and Y. If any element $x \in X$ can provide sub-stream $y \in Y$, then a connection element $e = (x, y)$ is added into connection set E. The corresponding weight of connection e is the hops of sub-stream y from stream source to provider x. We use the example in figure 3 to illustrate the construction of weighted bipartite graph. Neighbor p_1's available upload bandwidth can support 2 sub-streams, so there are 2 p_1 elements in the set X, as shown in figure 4. Neighbor p_1 can provide sub-stream 1, 2, 3 respectively, and the corresponding latency is 3, 4, 4 hops respectively. Accordingly, in the weighted bipartite graph, both of the two provider p_1 have connections with sub-stream 1, 2, 3 respectively, and the weight on each connection is 3, 4, 4 respectively. The whole weighted bipartite graph transformed from figure 3 is shown in figure 4. The number on each edge presents the weight of each edge.

3.3 Minimum delay, maximum matching algorithm (MDMMA)

After the weighted bipartite graph has been constructed, we try to find the best matching. Here we mainly consider two targets: first, more sub-streams should be transmitted in parallel, because this can make use of peers' upload bandwidth as much as possible, speeding up the transmission of the media content; second, hops of each sub-stream from source to the node should be as little as possible, because this can lead to lower delay. Existent matching algorithms of bipartite graph only care one of the two targets. In this paper, we enhance the $\mathtt{Hungarian}$ Algorithm which is the well known algorithm to solve the maximum matching of bipartite graph. The ameliorated $\mathtt{Hungarian}$ algorithm first insures that the matching is the maximum matching, and then it tries to find the lowest latency matching. The algorithm in detail is described in table 1.

The main changes occur in step 1.2 and step 1.4. In step 1.2, when there are several candidate "uncheck" providers, we choose the provider whose connected edge has the minimum weight. In step 1.4, to find the augment chain, we start from the vertex whose connected edge has the minimum weight, making sure that the edge with the lowest weight is put into the matching. Although the running time of the algorithm is $O(n^3)$, n is usually is less than 30, so the peer can find the appropriate neighbors quickly.

Table 1 Min Delay Max Matching Algorithm

Min Delay Max Matching

```
input   $G = (X, Y, E, W)$
output  $M$
0    set $M = \phi$ , and set all vertexes in $G$ with no label.
1.1  let $S = \{x/x \in X$, and exist $y \in Y$, $(x, y) \in M\}$. if $X/S = \phi$ , then finish;
     else label every elements $x \in X/S$ with "-1" and "uncheck".
1.2  if all $x \in X$ is checked, finish;
     else choose the "uncheck" $x_i$ whose connected edge has the minimum weight.
1.3  if all $y \in \{y/y \in Y$, and $(x_i, y) \in E\}$ have been labeled,
     then label $x_i$ with "checked", goto step 1.2.
1.4  let $P = \{y/y \in Y$, and $(x_i, y) \in E$, and $y$ is not labeled$\}$, label all $y \in P$ with "i".
     let $Q = \{y/y \in Y$, and exist $x \in X, (x, y) \in M\}$.
     if $P/Q \neq \phi$, then choose $y_j \in P/Q$ whose connected edge has the minimum weight, goto step 2;
     else for every $y_j \in P$, label $x_p((x_p, y_j) \in M)$ with "j" and "unchecke", label $x_i$ with "checked",
     goto step 1.2.
2    find the augment chain $C$ starting from $y_j$ until found $x(\in S)$ with label "-1", $M = M \oplus C$,
     cancel labels of all vertex in $G$, goto step 1.1.
```

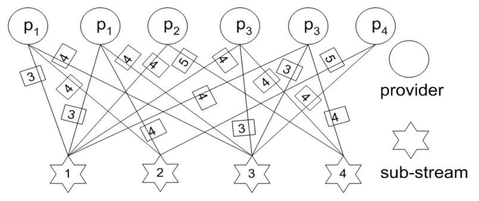


Figure 4: weighted bipartite graph

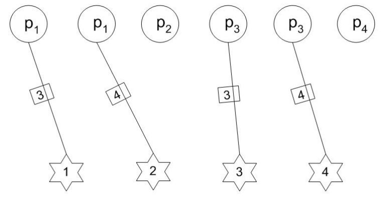


Figure 5: min delay, max matching

The ameliorated $\mathtt{Hungarian}$ algorithm is heuristic. It can insure that the matching is the maximum matching, but it can't insure that the matching is minimum delay matching theoretically. Applying the Min Delay Max Matching Algorithm to the weighted bipartite graph in figure 4, we get the following matching as shown in figure 5. According to the result, neighbor p_1 provide sub-stream 1, 2 to peer p, and neighbor p_3 provides sub-stream 3, 4 to peer p

respectively.

3.4 Broadcasting new sub-stream information

When a node decides to subscribe which sub-stream from which neighbor, it sends sub-stream request to the appointed neighbor. If the neighbor rejects the request, it cuts the matching connections in the weighted bipartite graph, and finds another provider with lowest latency. It requests again until it subscribe the sub-stream successfully or there is no useful connection. Then it broadcasts the subscribed sub-streams to its neighbors. The hop of each sub-stream in the information packets is adding the accepted hops by 1. When a neighbor accepts the request, it sends out the content blocks belonging to the desired sub-stream in the push window. If there is sub-stream(s) which can't find proper provider, the node should find more new neighbors.

3.5 Other discussion

Monitor neighbor's available upload bandwidth. We measure the interval of consecutive packets of the same sub-stream monitor neighbor's upload bandwidth. If the interval remains changeless (or change slightly), we suppose the bandwidth of the neighbor is affluent. If the interval changes greatly, we suppose the bandwidth of the neighbor is deficient. The rational reason behind this conclusion is that: if the bandwidth is affluent, there will no queue in the neighbors, little burst will happen, so the interval is changeless; if the bandwidth is deficient, burst will often happen, which leading to changing intervals.

Request missing blocks from neighbors with affluent bandwidth. Packets in transmission may be lost. When the missing content blocks enter into the pull widow, we request the block explicitly from the neighbor with affluent available upload bandwidth.

Sub-stream re-scheduling. Due to network dynamics, neighbors' upload capacity may fluctuate. When a neighbor is not competent as a sub-stream provider, a new provider should be chosen. Also we choose the appropriate provider from the neighbors with affluent. The neighbor who can provide sub-stream with fewer hops has higher priority.

4 Performance Evaluation

4.1 Simulation settings

We use the random model of GT-ITM [11] to generate the underlying topology with 5000 routers. Each link transmitting delay is set with a uniform random variable within [10ms, 100ms]. We randomly choose 2000 routers and connect one peer with one router. A peer randomly selects 10 to 15 other nodes as its neighbors to construct the mesh overlay. We set the playback rate of the original stream is 512kbps, and divide the stream into 32 sub-streams, each sub-stream with rate of 16kbps. The size of each content block is 1k bytes. The upload capacity of video source is 5 Mbps. All peers are assumed to be DSL users with three type of available upload bandwidth of 1 Mbps, 512 kbps and 256 kbps. These three types of peer represent 15%, 60% and 35% of the total peers. We suppose peer's download capacity is infinite. We run the simulation in the environment of Matlab7.0.

4.2 Simulation result

Here we compare our algorithm MDMMA with the pure pull-based method. We mainly care about the transmission delay of the stream. Figure 6 shows the cumulative distribution function of the content block transmission delay of the two methods in steady state. We can see that our

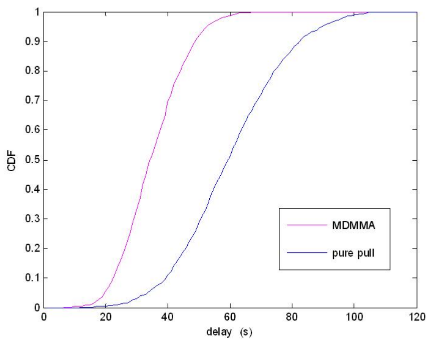


Figure 6: content block transmission delay CDF

algorithm MDMMA reduce the transmission delay greatly. For example, when 90% of the nodes get the content blocks, it only costs less than 50 seconds in our method, while in the push-based method, it costs more than 80 seconds. We can conclude that our proposed method is much better than pure pull-based method in the delay performance.

5 Conclusion

P2P streaming system consists of two components: (i) Overlay Construction Mechanism, and (ii) Content Scheduling Mechanism. Studies show that mesh-based system is better than tree-based system especially in high churn rate of node. In this paper, we present new scheduling mechanism to improve the performance of mesh-based P2P streaming systems. Our contribution includes two folds: first, we transform the sub-stream scheduling problem into the matching problem of the weighted bipartite graph; second, we present a minimum delay, maximum matching algorithm. Not only the maximum matching is maintained, but also the transmission delay of each sub-stream is as low as possible. Simulation shows that our method can reduce the transmission delay greatly.

Bibliography

[1] *PPlive*, http://www.pplive.com/.

[2] *Bittorrent*, http://bitconjuer.com/.

[3] X.Zhang, J.Liu, and et al. "Coolstreaming/donet: A data-driven overlay network for efficent media streaming". *In Proc. of INFOCOM 2005*, US, pp.2102-2111, Mar.2005.

[4] V.Pai, K.Kumar, and et al. "Chainsaw: Eliminating trees from overlay multicast". *Peer-to-Peer System ˘ô,* pp.127-140, Nov.2005.

[5] N.Magharei, R.Rejaie, and Y.Guo. "Mesh or multiple-tree: A comparative study of p2p live streaming services". *In Proc. of INFOCOM 2007*, USA, pp.1424-1432, May.2007.

[6] JM.Li, C.K.Yeo, and B.S.Lee. "Peer-to-peer streaming scheduling to improve real-time latency". *In Proc. of Multimedia and Expo*, China, pp.36-39, Jul.2007.

[7] M.Zhang, Y.Q.Xiong, and et al. "Optimizing the throughput of data-driven peer-to-peer streaming". *IEEE Transactions on Parallel and Distributed systems*, Vol.20, No.1,pp.97-110, May.2008

[8] M.Zhang, J.G.Luo, and et al. "A peer-to-peer network for live media streaming - using a push-pull approach". *In Proc. of the 13th annual ACM internatioan conference on Multimedia*, Singapore, pp.287-290, 2005.

[9] T.Locher, R.Meier, and et al. "Push-to-pull peer-to-peer live streaming". *In Proc. of DISC 07*, Germany, pp.388-402, 2007.

[10] Z.J.Li, Y.Yu, and et al. "Towards low redundancy push-pull P2P live streaming". *In Proc. of of ACM Sigcomm 2008 Demo*, USA, Aug. 2008.

[11] K.C.Ellen, W.Zegura and S.Bhattacharjee. "How to model an internetwork". *In Proc. of Infocom 1996*, USA, pp.594-602, 1996.