Packetization of 3D Progressive Meshes for Streaming over Lossy Networks

Yan Gu, Wei Tsang Ooi
Department of Computer Science
National University of Singapore, 117543

***Abstract*— In this paper, we consider the problem of packetizing progressive 3D geometry models for streaming over a lossy network. We adopt a graph-theoretic approach to model packetization of progressive 3D models, with the goal of minimizing dependencies among packets. We show that this packetization problem is strongly NP-complete, and we propose two packing heuristics. Our experiments show that both heuristics perform better than the naive method. In particular, the greedy packing heuristic produces significant improvement in the number of rendered nodes when the network is lossy.**

I. Introduction

Advancements in 3D rendering and scanning hardware have led to the increasing availability of 3D models. In this paper, we consider an emerging class of applications that require on-demand streaming and rendering of large-scale 3D models over the Internet, such as virtual museums and networked games. Similar to streaming of video and audio data, streaming of 3D data poses QoS challenges to applications, and has drawn the attention of network researchers. For example, issues such as unequal error protection [10], selective retransmission [1], and delivery to heterogeneous clients [7] have been considered in recent research.

In this paper, we consider the problem of packetizing 3D models for transmission over lossy networks. As in many previous studies, we employ progressive meshes [6] as the data model for network delivery. Encoding a 3D model as progressive meshes allows incremental rendering of the model with increasing details. Such encoding, however, introduces dependencies among vertices in the model, which may delay the rendering of a vertex if the packet containing the vertices it depends on is lost during transmission and has to be retransmitted. To minimize such delay, we should pack vertices into packets such that dependencies among packets are minimal. This problem can be modeled as a graph optimization problem called the acyclic equipartition problem. We show that this problem is NP-complete in a strong sense, which therefore does not admit FPTAS. As such, we adapt the subtree packing method from [5] to our packetization model. In addition, we propose an improved heuristic based on the subtree method for the packetization problem. We implement these algorithms and compare their performance in streaming progressive meshes over a network with different simulated loss rates. Our experiments show that for a dependency graph with 15150 nodes, the subtree algorithm can achieve up to 20% improvement in terms of number of renderable nodes at a time instance, compared to a breadth-first packing algorithm, and our greedy algorithm can achieve as much as 27% improvement over the breadth-first algorithm. Furthermore, we conduct the same experiments by transmitting progressive meshes between two nodes over PlanetLab and achieve similar performance improvement, indicating that our packetization algorithms are applicable and beneficial in the real network.

We briefly summarize the main contributions of our work as follows:

- We propose a packetization model for 3D progressive meshes and formulate the packetization problem as a graph optimization problem.
- We show that the packetization problem is NP-complete in a strong sense by reduction from 3-PARTITION. The proof is given in the previous paper [4]. Thus, no polynomial time algorithm exists unless P=NP.
- We design two heuristics to reduce dependencies among packets.
- We evaluate the performance of the two heuristics by streaming 3D models over LAN and PlanetLab. The results demonstrate that the heuristics achieve significant improvements in terms of rendering speed.

The remainder of the paper is organized as follows. Section II gives an overview of related work. We introduce the hierarchical 3D data model and investigate the packetization problem in Section III. In Section IV, we give our packing algorithms. We validate our heuristics in Section V. Section VI concludes our work.

II. Related Work

3DMC [8] specifies progressive meshes as one of MPEG-4's 3D model coding schemes. It encodes the topology data, connectivity and geometry information of a 3D model into a vertex graph (VG), triangle tree (TT) and triangle data (TD). VG contains the most important connectivity data while TT and TD contain geometry information. Different types of partitions are used for different important data. However, 3DMC does not consider dependency among partitions when they packetize a 3D model.

Our problem of acyclic equipartition is closely related to the well-studied graph partitioning problem, which is known to be NP-hard [3]. There have been attempts to solve these problems for the undirected graph with a polynomial time heuristic [2]. Wong has modified these algorithms and applied them to the directed graph to solve the acyclic multi-way partitioning problem with applications in VLSI design [9]; however, he has not considered equal-size partition in the work. Furthermore, the computational complexity of these algorithms is very high, and they are not suitable for real-time applications such as 3D meshes streaming.

The work closest to ours is Harris and Kravets' [5], which considers the problem of packetizing the bounding sphere tree for 3D models to possible subtrees, subject to the packet size constraint in breadth-first order. Since there is no correlation among sibling nodes in the tree, they do not consider inter-packet dependencies at all. In our work, we consider packetization of the dependency graph of progressive meshes, and explicitly consider dependencies among packets.

III. Data Model and Problem Definition

In this section, we briefly review what progressive mesh is and present a graph-theoretic model for our packetization problem.

A. Progressive Meshes

The progressive meshes scheme [6] is the most popular method to encode a 3D geometry model into different levels of details. It begins with the highest resolution model, which is defined as the *original model*. To get a lower resolution model, an operation called *edge-collapse* is performed to collapse an edge into a vertex. The edge-collapse operation is repeatedly performed until the model is reduced to its designated minimal resolution, which is referred to as the *base model*. A reversed operation, *vertex-split*, is executed during decoding. In vertex-split, a vertex v is split into two new vertices, v_1 and v_2. An example of vertex-split is shown in Fig 1. Given a base model and a series of vertex-split operations to be performed, we can completely reconstruct the original model. To deliver progressive meshes over a network, we would send the base model, followed by a series of vertex-split operations. We encode each vertex-split into a structure containing the vertices and faces involved in the operation.

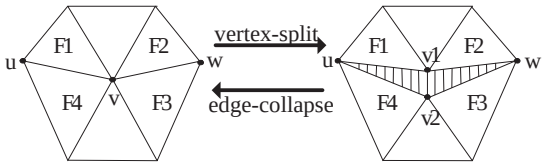

Fig. 1. Vertex-split and edge-collapse

Dependencies exist among vertex-splits. A face or a vertex needed in the current vertex-split operation might be the result of a previous vertex-split. If a packet p is lost, subsequent vertex-split operations that depend on the vertex-splits contained in p cannot be performed until p is retransmitted, resulting in rendering delay. It is therefore important to minimize dependencies among vertex-splits across packets, especially in a lossy network.

We can model dependencies among vertex-splits with a connected, directed acyclic graph $G = (V, E)$, called the *dependency graph*. Each node in V represents one vertex-split. An edge $(u, v) \in E$ exists if node v depends on node u. An example of the dependency graph is shown in Fig 2.

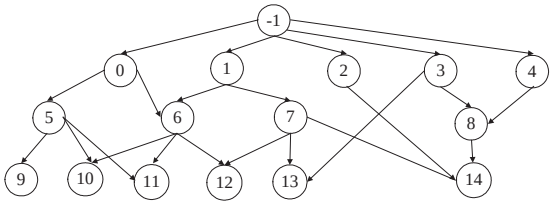

Fig. 2. Dependency graph

B. Problem Formulation

We will now define our packetization problem formally. Before we proceed with our problem formulation, we first define the notion of partition.

Given a dependency graph $G = (V, E)$, we can partition V into disjoint sets V_1, V_2, .. V_c, such that $\cup_{1 \le i \le c} V_i = V$ and $\cap_{1 \le i \le c} V_i = \phi$. Given any partitioning of G, we can construct a *partition graph* $G^P = (V^P, E^P)$, where $V^P = \{V_1, V_2, ..., V_c\}$, and $(V_i, V_j) \in E^P$ if and only if $(v_i, v_j) \in E$ for some $v_i \in V_i$ and some $v_j \in V_j$. We also define the *cut* of

a given partitioning as the set of edges whose incident nodes belong to different partitions. The number of edges in a cut is called the *cut size*. The intuitive meanings of these definitions are as follows: a partition models a packet, the partition graph gives the dependencies among packets, and cut size represents the degree of dependencies among the packets.

For convenience, we will use the terms partition and packet interchangeably in the rest of this paper.

We can now formally define our problem. *Given a connected directed acyclic graph $G = (V, E)$, $|V| = cB$ for $c \in Z^+$ and a bound $B \in Z^+$, find a partitioning of V into $\{V_1, V_2, V_3, ..., V_c\}$ such that the cut size is minimized, subject to the constraints that the resulting partition graph is acyclic and each partition is of equal size.*

We define any partitioning with equal size partitions and resulting in an acyclic partition graph as *acyclic equipartition*. The packetization problem defined above is called the CONNECTED ACYCLIC EQUIPARTITION (CAEP) problem. We prove that CAEP is NP-complete in a strong sense by reducing a known strongly NP-complete problem called 3-PARTITION [3] to CAEP in [4].

Theorem 1: CAEP is NP-complete in a strong sense.

IV. Packing Algorithms

In this section, we present our subtree packing algorithm and our greedy heuristic.

A. Breadth-first (BFS) Packing Algorithm

Due to the hierarchical structure of the dependency graph, it is reasonable for us to use the breadth-first search idea to achieve an evenly rendered quality in the 3D model. By rendering the model level by level, we would guarantee the resolution increases equally throughout the model.

The BFS algorithm in this paper extends the basic breadth-first search strategy by checking the parents of the traversed node u that have already been packed either in the same partition or in the precedent partitions such that packing u does not incur any cycle in the partition graph. The algorithm traverses the dependency graph level by level, and does not aim to minimize dependency at all.

B. Breadth-first Subtree (BSub) Packing Algorithm

We extend the subtree idea from [5] to pack the dependency graph, which is presented as *BSub*. First, we put the $root$ of the graph into a FIFO queue. For every new packet, we pick one node from the queue as the root of the new packet and traverse the subgraph of this node. Once we find the child in the subgraph whose parents have been packed, we pack this child until the packet is full, then we put the child into the queue.

Before we present the pseudo code of BSub, we first define some data structures and variables in the BSub algorithm. Variable B is the total number of nodes per packet. Variable $root$ is the root node of dependency graph G. Variables Q_t, Q_p, Q_r are FIFO queues in which each element is one node of G. Queue Q_r contains each new packet of B nodes; the last packet may contain less than B nodes in case $|V|$ is not divisible by B. Queue Q_p contains the nodes with packed parents. Queue Q_t contains the nodes dumped from Q_p when Q_r is full. For each full packet, GenPkt(Q_r) is called to generate a new packet for elements in Q_r and initialize Q_r to empty. The function Dump(Q_p,Q_t) removes every element from Q_p to the tail of Q_t. The sets $child[u]$ and $parent[u]$ for $u \in V$ contain the children of u and the parents of u respectively. The function Enqueue(Q, x) inserts the node x into Q and Dequeue(Q) removes the head of Q. The function Ready(x) returns TRUE if all parents of node x have been packed in previous packets or returns FALSE otherwise. The BSub algorithm is shown below.

1 BSub($root$)

```
1:  Q_t ⇐ ∅
2:  Q_p ⇐ ∅
3:  Enqueue(Q_t,root)
4:  while |Q_t| > 0 do
5:     while |Q_r| < B and (|Q_p| > 0 or |Q_t| > 0) do
6:        if |Q_p| > 0 then
7:           w ⇐ Dequeue(Q_p)
8:           Enqueue(Q_r,w)
9:           for all u ∈ child[w] do
10:             if Ready(u) then
11:                Enqueue(Q_p,u)
12:             end if
13:          end for
14:       else if |Q_t| > 0 then
15:          w ⇐ Dequeue(Q_t)
16:          Enqueue(Q_p,w)
17:       end if
18:    end while
19:    GenPkt(Q_r)
20:    Dump(Q_p,Q_t)
21: end while
```

C. Greedy Subtree (GSub) Packing Heuristic

The BSub packetization algorithm does not seek to minimize dependencies. In this section, we describe a

heuristic which greedily packs nodes such that the local cut size is minimized. We term this new heuristic *GSub*.

In the GSub algorithm, we adopt the subtree idea from BSub. Furthermore, every time we consider one child to be packed from a subgraph, we choose the child whose parents have been packed and which introduces the minimal cut size as well. We maintain a max-priority queue of children. After we pack a packet, we update the key value $key[u]$ for each child $u \in V$, where $key[u] = |parent_{in}[u]| - |child[u]|$.

Fig 3 illustrates the general case of the greedy picking strategy where node u and node v are both non-leaf nodes. V_1 and V_2 are two packets. We assume V_1 is fully packed before the current packet V_2. The sets $parent_{out}[u]$ and $parent_{in}[u]$ for $u \in V$ contain the parents of u which are not in the current packet V_2 and which are within V_2 respectively. Thus, we have $|parent_{out}[u]| + |parent_{in}[u]| = |parent[u]|$. The current cut is $cut = |parent_{out}[u]| + |parent_{in}[u]| + |parent_{out}[v]| + |parent_{in}[v]| = |parent[u]| + |parent[v]|$. The cut after we pick node x is updated as $cut^{'} = |parent[u]| + |parent[v]| - (|parent_{in}[x]| - |child[x]|)$. To minimize $cut^{'}$, we maximize $|parent_{in}[x]| - |child[x]|$.

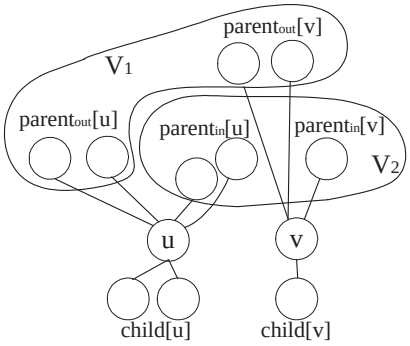

Fig. 3. GSub greedy packing: pick the first node from two non-leaf nodes u and v. Since $|parent_{in}[u]| - |child[u]| = |parent_{in}[v]| - |child[v]| = 0$, we randomly pick any u or v. $cut^{'} = 4 + 3 - 0 = 7$.

In the pseudo code of GSub, H_p and H_t are defined as max-priority queues implemented with heap. The function Extract-Max(H) removes and returns the element of H with the largest key. The function Insert(H,u) inserts node u into H, maintaining the max-priority property. The other structures and functions shown in GSub follow the same definitions as in BSub.

D. Time Complexity

The complexity of BSub is $O(|V| + |E|)$. For GSub, since the complexity of Extract-Max(H) and Insert(H,u) is $O(\lg |V|)$, the running time of GSub is $O(|V| + |E| \lg |V|)$ in the worst case.

2 GSub($root$)

```
 1: H_t ⇐ ∅
 2: H_p ⇐ ∅
 3: Insert(H_t,root)
 4: key[root] ⇐ −|child[root]|
 5: while |H_t| > 0 do
 6:    while |Q_r| < B and (|H_p| > 0 or |H_t| > 0) do
 7:       if |H_p| > 0 then
 8:          w ⇐ Extract-Max(H_p)
 9:          Enqueue(Q_r,w)
10:          for all u ∈ child[w] do
11:             if Ready(u) then
12:                key[u] ⇐ |parent_in[u]| − |child[u]|
13:                Insert(H_p,u)
14:             end if
15:          end for
16:       else if |H_t| > 0 then
17:          w ⇐ Extract-Max(H_t)
18:          Insert(H_p,w)
19:       end if
20:    end while
21:    GenPkt(Q_r)
22:    while |H_p| > 0 do
23:       w ⇐ Extract-Max(H_p)
24:       Insert(H_t,w)
25:    end while
26: end while
```

V. Performance Evaluation

A. Experiment Setup

We set up our experiment on a 1 Mbps LAN connecting two Redhat 9.0 Linux servers. In our experiment, we encode a 3D model as a base model and a sequence of vertex-split structures which are modeled in a dependency graph. In the sender, we apply the BFS or BSub or GSub packing algorithm to the dependency graph on the fly. We pack 18 vertex-splits into one packet according to the constraint of Maximal Transmission Unit (MTU). We use TCP to transmit the base model as it is required for the decoding of sequential vertex-split structures. The vertex-splits are transmitted with UDP. Lost packets are retransmitted. In the experiment, we simulate different network scenarios with different packet loss rates ranging from 2% to 10%.

In addition, to verify the applicability and performance of the packetization algorithms over wide-area network, we stream 3D models from Berkeley to Hong Kong over

TABLE I
CUT SIZE OF DEPENDENCY GRAPH GENERATED BY DIFFERENT PACKETIZATION ALGORITHMS

model	$\|V\|$	$\|E\|$	*B*	*BFS*	*BSub*	*GSub*
goblet	496	1353	18	1350	1054	1007
epcot	764	1778	18	1777	1306	1189
sphere	3333	4468	18	4465	2036	1626
horse	15149	10582	18	10570	2845	2297
horse	45149	81244	18	81244	44112	36623
horse	48149	98813	18	98798	58652	50384

PlanetLab and measure the loss rate between the two nodes in addition to the rendering speed.

When the user receives a packet, it decodes all packets in the rendering buffer. Any renderable nodes are decoded at once. Due to the dependency among nodes in the dependency graph, some nodes may not be decoded as their ancestors may not be in the rendering buffer at that time, which accounts for the increment in cut size would delay the decoding of nodes.

B. Cut Size

To evaluate the performance of GSub and BSub, we first compare the resulting cut sizes of different models with different levels of refinement. The results are shown in Table I. GSub produces a smaller cut size than BSub, and achieves a large improvement over BFS.

C. Number of Rendered Nodes

In our experiment, each time we receive a new packet, we decode all available nodes and count the total number of decoded nodes. In an ideal network without any packet loss, the cut sizes of different packing algorithms are irrelevant to the quality of the 3D model. However, when packets are lost and need to be retransmitted, the difference in cut size of different packing algorithms will have a significant impact on rendering quality. Hence, we measure the number of rendered nodes at a certain moment to evaluate the performance of different packing algorithms.

Fig 4 to Fig 6 show the number of rendered nodes with the simulated loss rate of 2% to 10% for a horse model with 15150 vertex-splits. Each figure compares the three algorithms introduced in Section IV. All nodes packed with three different algorithms take the same amount of time to finish rendering on account of the same total nodes transmitted with the same speed. The differences in the number of rendered nodes during transmission demonstrate GSub minimizes the delay and achieves a best user experience among the three algorithms.

Fig 7 shows the improvement of BSub and GSub over BFS with the simulated loss rate. We measure the improvement of BSub over BFS using the maximal difference in the number of rendered nodes between BSub and BFS over BFS. The improvement of GSub over BFS is measured using the maximal difference in the number of rendered nodes between GSub and BFS over BFS. We can see the improvement of BSub increases from 3% to 20% when the loss rate rises from 2% to 10%. The improvement of GSub increases from 5% to 27% when the loss rate increases from 2% to 10%. GSub outperforms BSub in terms of the number of rendered nodes as well. Meanwhile, we notice that the superiority of GSub over BSub is not so significant as that of GSub over BFS because a considerable amount of dependencies are reduced when packing the children with their parents from the same subgraph in BSub.

In our experiments over wide-area network, we achieve similar performance improvement. Fig 8 compares the packetization algorithms over the PlanetLab nodes from Berkeley to Hong Kong with a loss rate of 18%. In order to measure the results with the same loss rate, we repeat the experiments for three packetization algorithms and pick the ones with loss rate of 18% for comparison. The improvement of BSub over BFS is 30% while the improvement of GSub over BFS is 44%.

VI. CONCLUSION

We have presented a dependency-aware packetization model for hierarchical progressive meshes to speed up the rendering of 3D models delivered over a lossy network. We adapt the BSub packing method to our dependency graph and achieve a better cut size and an improved number of rendered nodes over BFS. To reduce the cut size, we propose the GSub packing heuristic. Evaluations show that the GSub algorithm works well compared to BSub in terms of number of rendered nodes when the network is lossy.

ACKNOWLEDGMENT

We would like to thank Tran Hong Quang and Kaicheng Zhang for their help during the early stage of this research, and the reviewers for their valuable suggestions.

REFERENCES

[1] C. L. Bajaj, S. Cutchin, V. Pascucci, and G. Zhuang. Error resilient transmission of compressed VRML. Technical report, University of Texas at Austin, 1998.

[2] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal of Algebraic Discrete Methods*, (3):541–550, March 1982.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[4] Y. Gu. Dependency-aware packetization problem in progressive meshes. In *IEEE Infocom (Poster)*, Miami, FL, USA, March 2005.

[5] A. F. Harris, III and R. Kravets. The design of a transport protocol for on-demand graphical rendering. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 43–49, Miami, Florida, USA, 2002. ACM Press.

[6] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 99 – 108, New Orleans, Louisiana, USA, August 1996.

[7] I. M. Martin. Hybrid transcoding for adaptive transmission of 3D content. In *IEEE International Conference on Multimedia and Expo*, pages 373–376, Lausanne, Switzerland, August 2002.

[8] F. Pereira and T. Ebrahimi. *The MPEG-4 Book*. Multimedia. IMSC Press and Prentice Hall, 2002.

[9] S. H. Wong, F. Y. Young, and W. K. Mak. Clustering based acyclic multi-way partitioning. In *Proceedings of ACM Great Lakes Symposium on VLSI*, pages 203–206, Washington, DC, USA, April 2003.

[10] Z. Yan, S. Kumar, and C.-C. J. Kuo. Error resilient coding of 3D graphic models via adaptive mesh segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(7):860–873, Jul. 2001.

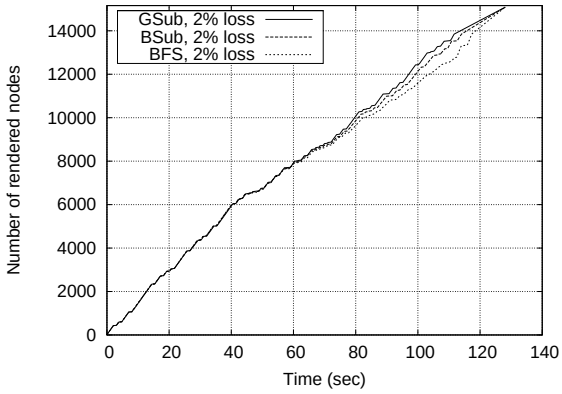

Fig. 4. Comparison in number of rendered nodes among BFS, BSub and GSub with the simulated loss rate 2%

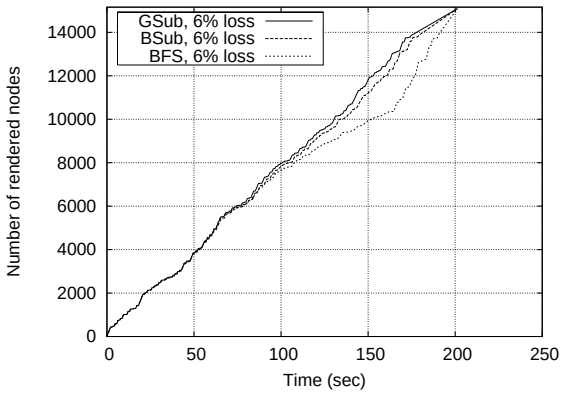

Fig. 5. Comparison in number of rendered nodes among BFS, BSub and GSub with the simulated loss rate 6%

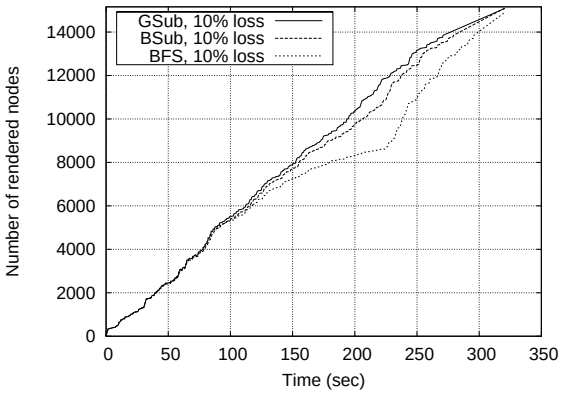

Fig. 6. Comparison in number of rendered nodes among BFS, BSub and GSub with the simulated loss rate 10%

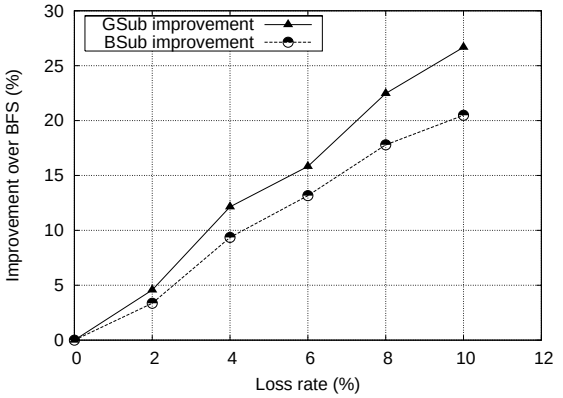

Fig. 7. Improvement of BSub and GSub over BFS

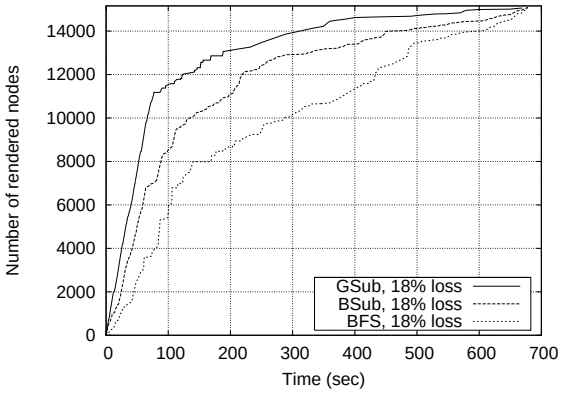

Fig. 8. Comparison in number of rendered nodes among BFS, BSub and GSub over PlanetLab network