

Efficiently Maintaining Consistency Using Tree-Based P2P Network System in Distributed Network Games

Kyung Seob Moon Vallipuram Muthukumarasamy Anne Thuy-Anh Nguyen

School of Information and Communication Technology, Griffith University, Australia
k.moon@griffith.edu.au v.muthu@griffith.edu.au a.nguyen@griffith.edu.au

Abstract

There are two main approaches, conservative and optimistic, for maintaining consistency in distributed network games. Under the conservative approach, players may experience network latency, depending on packet transfer delay caused by the send-and-wait and acknowledging processes. Under the optimistic approach, the processes do not wait for other players' packets and advance to their own frames, hence there is no network latency. However, when inconsistency happens, the processes must roll back. This can cause irritation and confusion to players, and thus the game quality deteriorates. Overall, the optimistic approach may not be suitable for networked games. To overcome the network latency problem in the conservative approach, we propose a new system which can reduce the network latency and bandwidth requirements. Furthermore, the effect of the number of players in multiplayer game sessions is examined in detail with varying number of players. Experimental results with our proposed system confirm improved performances in latency and frame rate.

Keywords: Consistency Maintenance, Distributed Games, Interactive Games, Low-latency, Peer-to-Peer System

I. INTRODUCTION

The estimated value of the worldwide gaming market was over US \$30 billion in 2002 and has been growing rapidly. This, in fact, exceeded the size of the movie industry [1]. The shareware computer game *Doom* was one of the most popular games in computer game history and 15 million copies have been downloaded

around the world. The 3D environment and the various modes of networking play such as co-operative and death-match modes have been cited as factors contributing to the game's popularity [2]. Since the great success of *Doom*, the majority of games feature network play functionality, and network games have become more and more popular, mainly because of the increased strategic complexity of network play [3].

Multiplayer Computer Games (MCGs) prefer, in general, Client/Server (C/S) network architecture to Peer-to-Peer (P2P) system. This is mainly because of its advantages, such as simplicity of consistency maintenance, better security, authentication, and easy billing system. However, C/S architecture can cause network latency and servers may become the network bottleneck [4]. To solve this problem, server clustering methods are used, however they may not be cost effective solutions. Other types of games use P2P network architecture but the total number of players in one game session is limited because of the networks' bandwidth problem. In addition, the consistency maintenance issue needs to be addressed in P2P architecture.

In C/S structure, clients send packets which consist of object states or user commands to the server. Subsequently, the server determines the validity of the packets and sends back the result to the client. The states of all objects in the game are kept on the server side, so there is almost no inconsistency issue compared to the P2P structure. On the other hand, each node manages its own object states in the P2P structure, thus inconsistencies invariably occur due to network delay.

As mentioned above, C/S structure can cause network latency because of the two-way packet transfer which is from Client to Server and then from Server to Client. So, if T is the network latency between Server and Client then $2T$ is required for communication. Whereas, only T is necessary in P2P structure because of the fact that movement validations are made in each peer. If P2P uses unicast and unstructured overlay then each node sends the same packet to all other peers and this creates network bandwidth problems. This may severely constrain the maximum number of players.

In this paper, we propose a tree-based P2P network system which addresses the P2P network latency and bandwidth problems. Experiments are carried out with 4, 8, and 16 player game sessions and the results are analysed. In the next section, related work in the consistency maintenance area is briefly reviewed. In Section III, the basic concept and architecture of Tree-Based P2P system is introduced. Experimental details and results are described in Section IV. Analysis and discussion are presented in Section V. Conclusions are given in Section VI.

II. RELATED WORK

The Lock-Step algorithm [5] is one of the simplest solutions for consistency maintenance in the P2P structure. Each peer waits for other peers' packets of the current frame, makes its next move, sends packets and waits again. The drawback of this approach is that it can cause slowdown for game play if network latency is slower than the frame rate. For example, if the game's FPS (Frame per Second) is 25 then each frame takes about 40ms to load. In case the network latency is longer than 40ms then players will have to wait until they get other players' packets.

The Frequent State Regeneration [11] approach eliminates the slowdown-time of the Lock-Step algorithm by frequently transmitting the status of objects in game sessions. Generally, an unreliable protocol such as UDP is used with this approach to alleviate the overhead of using a reliable protocol such as TCP. However, sending the status of objects frequently to all players requires high bandwidth and this requirement limits the maximum number of players for network games.

The Bucket Synchronization algorithm [6] and Local Lag [8] introduce artificial delay so as to synchronize a peer's own frame with other nodes' frame by utilizing

imperfect human visual perception. This approach is analogous to the buffering method of streaming audio. Even though the lock-step time is extended, it still requires inconsistency resolution algorithms because network latency is longer than the extended lock-step time. Bucket Synchronization and Local Lag also require high bandwidth due to the frequent state transmission inherent in their solutions.

Dead Reckoning [7] algorithms interpolate and/or extrapolate missing and/or incoming information to reduce bandwidth requirement and latency. The Local Perception Filter [12] approach also utilizes the limitation of human eye perception by altering the speed of objects in networking games to mask network latency. The Time Warp [9] algorithm has been introduced to solve inconsistency and/or network latency problems by adapting optimistic approaches. However, the overhead of rollover process is unavoidable.

III. TREE-BASED P2P NETWORK SYSTEM

To maintain fairness for all players in the game session, the game speed is synchronized with the slowest process and the network latency node. There are two variables to be considered: process speed and network latency. The first variable, process speed, is easy to test before game sessions start and it is not changed dramatically during game sessions. Therefore, process speed problem can be solved by pre-testing whether players' computers can process adequate frame rates during game sessions. However, network latency is hard to predict accurately and it changes during game sessions. In this paper, we have proposed a tree-based P2P system to alleviate some of the problems.

A. Graph to tree conversion

Existing P2P games use a graph structure to form a network topology, where each node connects to all other nodes. As explained before, it can cause bandwidth problems if unicasting is used. Furthermore, since all nodes are synchronized with the slowest node, this structure cannot take advantage of fast connections between any other nodes.

On the other hand, if network topology is structured by tree-based architecture then nodes will be hierarchical and fast connection nodes will be grouped together. Figure 1(a) shows the graph structure and network latency values between nodes. Figure 1(b) displays the final result after removing high network

latency links from the graph structure. Node A creates its own tree structure as shown in Figure 1(c).

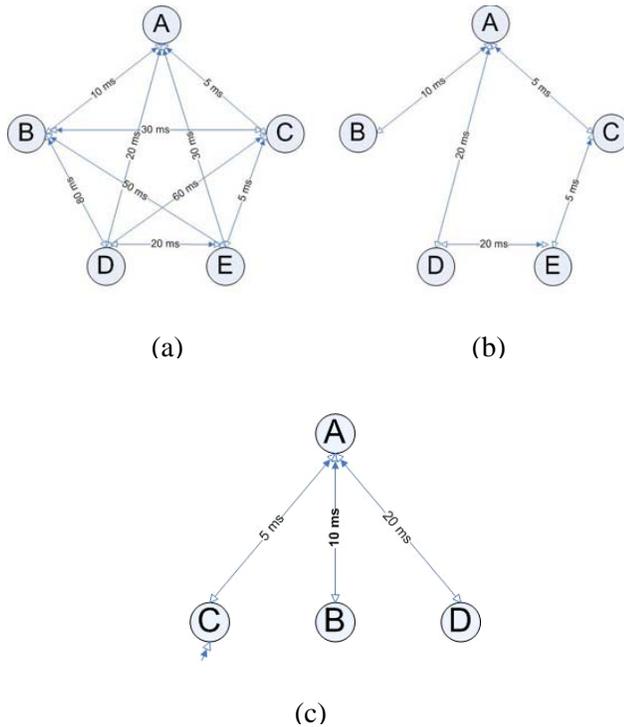


Fig 1. Finding alternative paths for high network latency nodes and conversion from graph structure to tree structure for node A.

As can be seen in Figure 1(a), the longest path from node A to other nodes is the link between A and E which takes 30 ms. We can utilize the A – C – E path with the total time from node A to E of 10 ms. The new path reduces network latency by 20 ms.

Applying this process to paths between all nodes can increase overall game speed by reducing the network latency of the slowest link in the network topology. As it can be seen from Figure 1(a), since the B-D connection is the slowest link and the B-A-D link can reduce network latency by 50 ms. For graph to tree conversion, one of the most well known shortest path finding algorithm, Dijkstra algorithm [10], is used.

B. Relay Method

Nodes in this tree-based P2P system may need to relay packets to its child nodes because sender nodes may not have direct connection to destination nodes. In the relay process, the sender node becomes a root node and transmits packets to its own child nodes which are only one level below them. The sender node includes

the identification numbers (ID) of destination nodes in its packet header. When its child nodes receive the packet they remove their ID from the packet header and relay this packet to their own child nodes.

Figure 2 shows the tree structure for each node. When node A sends a packet to other nodes, it includes the IDs of destination nodes. To send a packet From A to E, the destination node list will be [C, E]. When node C receives the packet from node A, it removes itself from the destination node list and relays the packet to node E only because node A is not in the destination node list. The pseudocodes below show how the relay algorithm works.

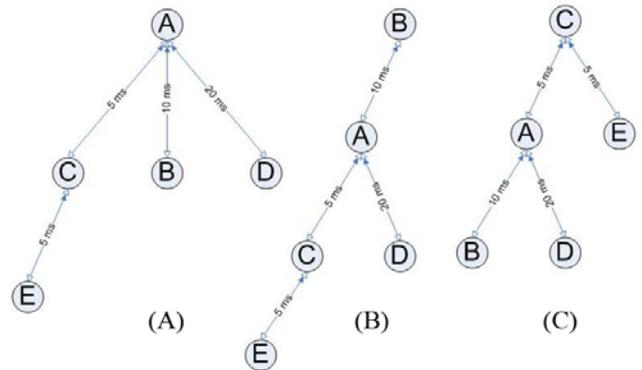


Fig. 2. Tree Structures for each node (Node A, B, and C only)

```

/*Pre-condition: nodeNum: A
  origin node list: originNodeList
  destination node list: destNodeList
Post-condition: packets are relayed to immediate child
nodes */
// Frame Packet Relay Algorithm
relayFramePacket ()
  tempDestNodeList = [ ];
  add A into originNodeList;
  delete A from destinationNodeList;
  for each node in destinationNodeList C
    recursively visit each node in
    destinationNodeList
      if each node is in destinationNodeList
        add the node into tempDestNodeList
      end if
    send a packet with tempDestNodeList and
    originNodeList to C;
  end for
end relayFramePacket;

```

C. Retransmission

Retransmission is unavoidable when packet drop or errors occur, especially when frequent packet regeneration scheme [11] is not used. The decision for

retransmissions is based on ACK timeout and ACK numbers from opponent nodes. The retransmission procedure is exactly the same as sending packets the first time except that the retransmission will occur at the parent node of the node that requests the missing packets.

D. Acknowledgement

This tree based P2P system can prevent ACK implosion problem because, each node manages its own child or children. When a packet is sent, the destination node sends ACK to its source node immediately. The sender retransmit the packet if $(ACK_timeout_of_destination_node < Current_Waiting_Time_for_ACK)$ is true. The ACK timeout is based on ping time between the two nodes.

E. Simulator Architecture

The tree-based P2P network simulator is implemented in C++ with STL and consists of three main classes, namely, 1) Player, 2) Simulator and 3) Statistics. The Player class utilizes three data structure types: Queue for an input buffer to store other players' packets, Priority Queue for a re-sending buffer to re-send dropped packets, and Circular Array for a game buffer to gather frame data. Figure 3 shows the basic architecture of the Simulator. The Simulator class is responsible for packet forwarding among players by providing MainBuffer, packet dropping by applying packet drop rate, and passing simulation results to the Statistics class which records the data into files for later data analysis. The Player class is in charge of packet relay, sending acknowledgement, game frame packet generation and packet resending when packet drop is detected.

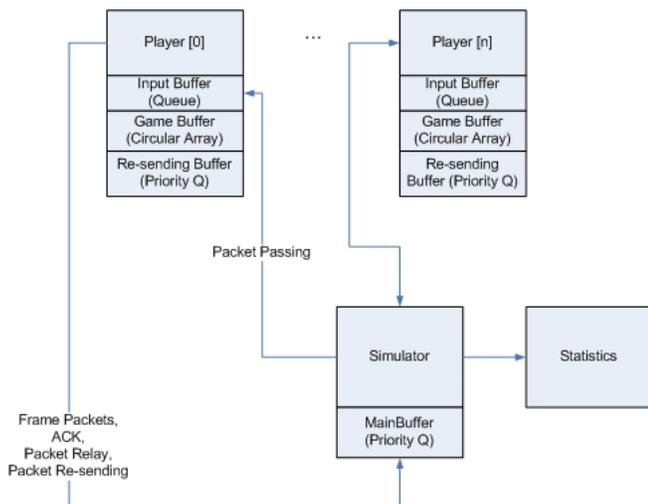


Fig 3. Tree-based P2P Network Simulator Architecture

IV. EVALUATION OF EXPERIMENTAL RESULTS

The tree-based and graph-based P2P networks are evaluated and compared in this section. An Intel Pentium 4, with 1.60GHz CPU and 512MB RAM, equipped machine with Microsoft Windows XP operating system was used for all the experiments.

A. Experimental Data Set

In general, most of network games support a maximum of eight players in a P2P network, due to constraints such as latency and bandwidth. Therefore, we decided to make the simulator with generated data sets of four, eight and sixteen players with randomly selected network latency with values between 5 and 100 ms. Table 1, 2 and 3 show maximum and average latency between each node. The tables also present travel distance between each node in milliseconds except in Table 3, due to space limitation. Maximum latency is a key factor which affects the overall game speed. The maximum latency of the four, eight, and sixteen-player experimental data set are 82, 77, and 100 respectively. We will explain the meaning of maximum latency further in the next section.

TABLE 1
LATENCY VALUES BETWEEN FOUR PLAYERS
Max Latency: 82, Average Latency: 46.83

Player	0	1	2	3
0	0	5	59	23
1	5	0	82	61
2	59	82	0	51
3	23	61	51	0
Max:	59	82	82	61
Avg.:	41	43	44.43	31.57

TABLE 2
LATENCY VALUES BETWEEN EIGHT PLAYERS
Max Latency: 77, Average Latency: 40.46

Player	0	1	2	3	4	5	6	7
0	0	26	58	76	10	46	11	60
1	26	0	64	52	37	58	36	28
2	58	64	0	15	48	77	35	54
3	76	52	15	0	18	15	21	24

4	10	37	8	18	0	27	77	25
5	46	58	77	15	27	0	76	9
6	11	36	35	21	77	76	0	70
7	60	28	54	24	25	9	70	0
Max:	76	64	77	76	77	77	77	70
Avg.:	41	43	44.4	31.6	34.6	44	46.6	38.6

TABLE 3
LATENCY VALUES BETWEEN SIXTEEN
PLAYERS

Max Latency: 100, Average Latency: 52.10

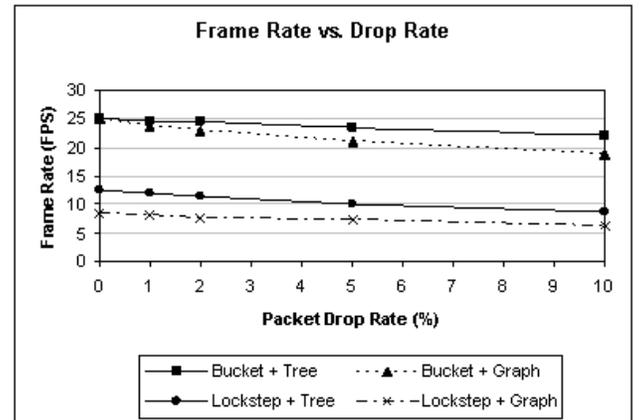
Player	0	1	2	3	4	5	6	7
Max:	91	99	89	100	96	93	96	100
Avg.:	55.87	32.80	52.47	60.93	42.40	60.07	55.67	48.47
Player	8	9	10	11	12	13	14	15
Max:	91	96	97	89	96	89	100	100
Avg.:	47.13	59.67	51.27	41.67	54.07	50.20	58.13	62.80

B. Comparison between Graph-based and Tree-based structure in three data sets

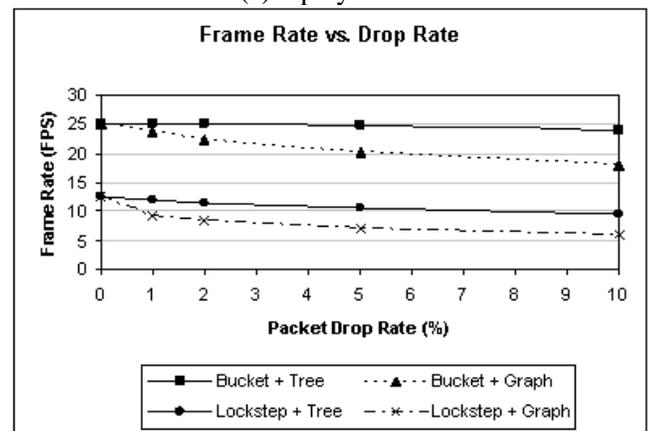
Frame rate and average latency were recorded on four different systems which are combined using two consistency maintenance algorithms (Bucket Synchronization, and Lock-Step) and two network types (Graph-based, and Tree-based). These processes were repeated to collect experimental results from three data sets stated in Section 4-A.

Figure 4 shows the changes in frame rates with respect to the packet drop rate in the four systems on the game sessions of 4, 8, and 16 players. When the packet drop rate is 0, the frame rates of Lock-Step with Graph-based system are 8.35, 12.51 and 8.35 for the number of players 4, 8, and 16 respectively. It is mainly related to frame interval and maximum latency. Current frame interval is 40ms and the maximum latency of the four-player-game session is 82ms. Because the Lock-Step algorithm holds the game process until the arrival of all packets from other players and frame packet transmission occurs at every 40 ms, every player must wait for the maximum latency, which is 82ms, and sends packets at every 120 ms. These characteristics of the Lock-Step algorithm explain the frame rates of Lock-Step with Graph-based system under the condition of no packet drop for each case. On the other hand, the Bucket-Synchronization algorithm lets players send their frame packets without waiting for other players' packets until the latency of other packets is less than the playout delay which is 120 ms in this occasion. The maximum latency values of three data

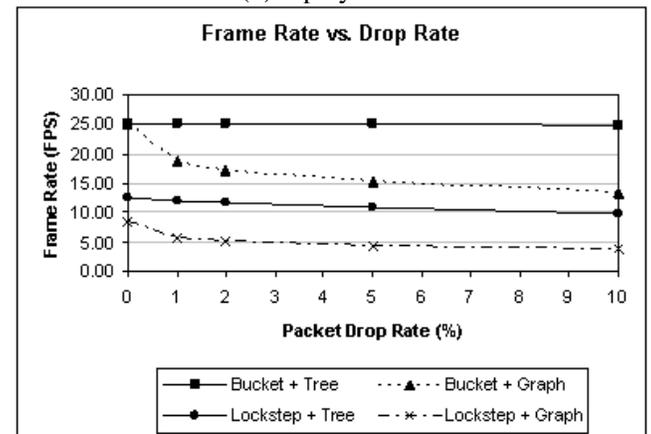
sets are all less than or equal to 100 ms, thus the optimal frame rate value is determined as 25 fps.



(a) 4 players



(b) 8 players

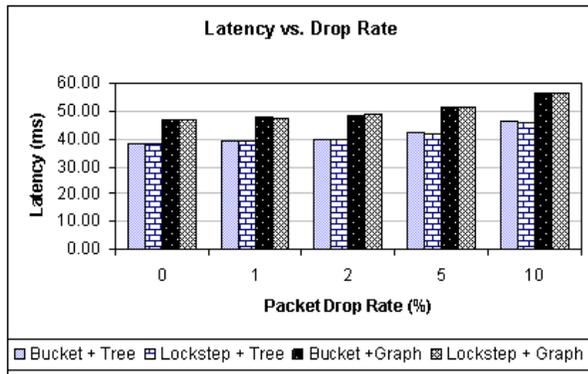


(c) 16 players

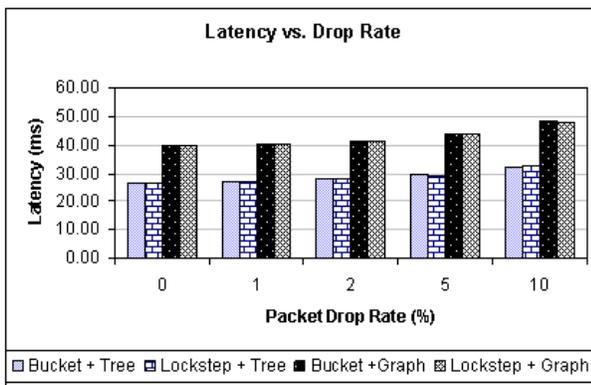
Fig. 4. Frame Rate vs Drop Rate (4, 8, and 16 players)

The graphs for the average packet propagation time with packet drop rate on the four combinations are shown in the figure 5. As can be seen from the figures, the latency of the two Graph-based approaches is similar and so is the latency of the two Tree-based approaches. The Graph to Tree conversion improves

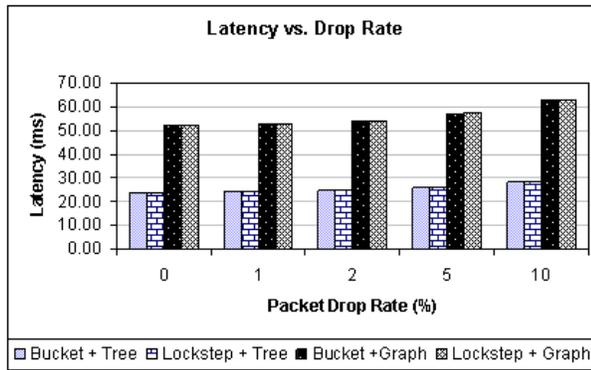
the network delay from 46.82 to 38.33ms, from 39.74 to 26.93ms, and from 52.09 to 23.51ms in 4, 8 and 16 player-game sessions respectively.



(a) 4 players



(b) 8 players

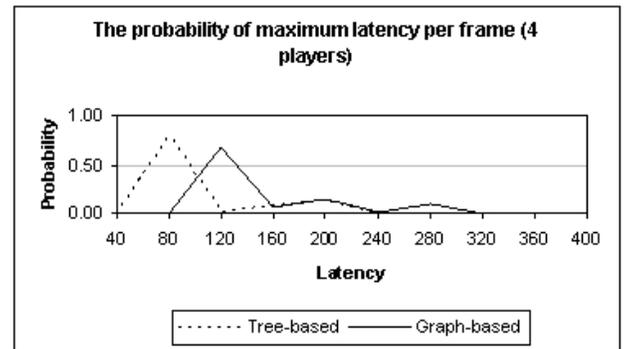


(c) 16 players

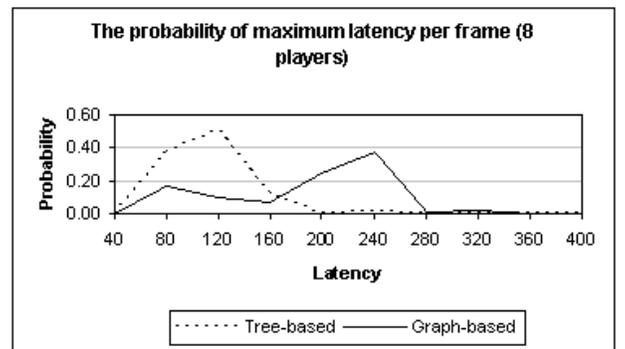
Fig. 5. Latency vs Drop Rate (4, 8, and 16 players)

Figure 6 shows the probability of maximum frame latency in each frame on two cases, Graph-based and Tree-based P2P network systems, with the Bucket Synchronization approach. The experimental parameters are 5% packet drop rate, 120 ms playout delay and 40 ms frame interval. The average incidence of delayed frames was reduced from 9.24% to 3.02%, from 14.58% to 1.58%, and from 37.81% to 0.12%

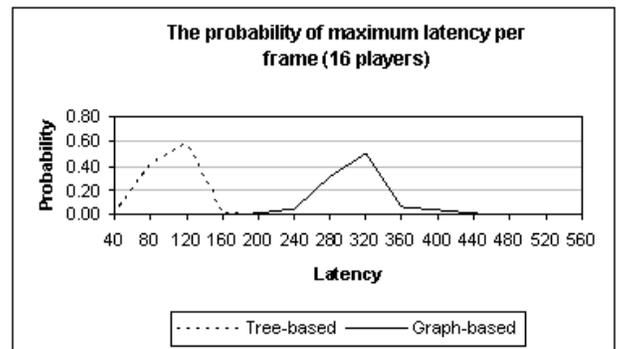
with Graph-based P2P network and Tree-based P2P network in 4, 8 and 16 player-game sessions respectively.



(a) 4 players



(b) 8 players



(c) 16 players

Fig 6. The probability of maximum latency in each frame on the two approaches with parameters 5% drop rate, 120 ms playout delay and 40 ms frame interval.

Figure 7 shows the percentage of improvement in frame rate in Bucket-Synchronization with the Tree-based approach. The values are defined as the ratio of improvement of frame rate with respect to the frame rate of the tree-based P2P network system with four-player-game session. According to Figure 4, when the number of players increases, frame rate decreases

drastically in Bucket-Synchronization with Graph-based approach. However, as can be seen in Figure 7, the Bucket-Synchronization with Tree-based approach does not degrade the frame rate when the number of players increases, rather it performs better.

The percentage of improvement in average latency is displayed in Figure 8. The figure shows that the average latency is improved as the number of players increases. Also, packet drop rate has no effect on the percentage of improvement in average latency.

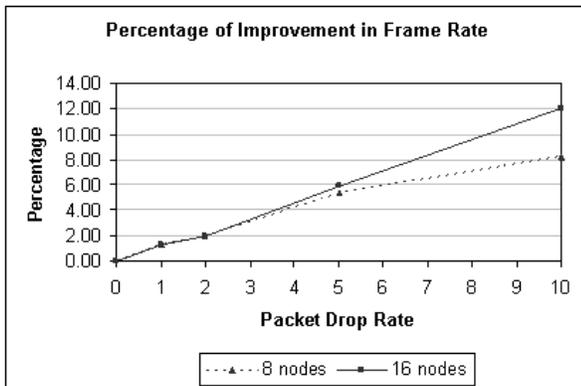


Fig. 7. Percentage of Improvement in Frame Rate

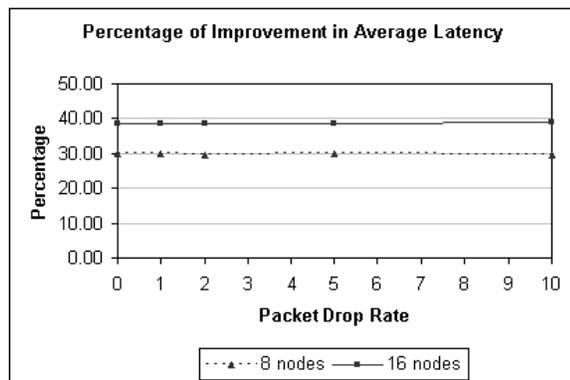


Fig 8. Percentage of Improvement in Average Latency

V. ANALYSIS AND DISCUSSION

As can be seen from Figure 4, the frame rate of game sessions drops significantly when the number of players increases in Graph-based network systems. On the other hand, the change rate of the frame rate of game sessions becomes lower as the number of players increases in Tree-based network systems as is evident from Figure 7. *This result may stem from the fact that a greater number of players necessarily leads to a higher number of potential paths over which to optimize.*

Figure 5 also shows similar effects on the number of players on network latency. According to the graphs, network latency is reduced more as the number of players increases in Graph to Tree conversion. The same is also evident from Figure 6. The shape of the Tree-based diagram is similar to the Graph-based diagram in each session. *We postulate that this is because the reduced network latency is the sole parameter that is changed in this experiment. The more the number of players, the greater the distance of parallel displacement from the chart of Graph-based system. Also, the distance is related to the reduction rate of network latency.*

VI. CONCLUSION

In this paper, we have evaluated the efficiency of the Tree-based P2P approach by comparing it with the Graph-based P2P approach in the game sessions of various numbers of players. Two consistency maintenance algorithms, Lock-Step and Bucket Synchronization, were implemented to analyze the efficiency of the Tree-based approach in three types of game sessions, which are small, medium and large sized P2P game sessions. Our experimental results showed that the Tree-based approach with the Bucket Synchronization algorithm performed at almost optimal frame rate, between 24 and 25 fps, even with a 10% packet drop rate. Also, the increase of the number of players does not degrade the efficiency of the game sessions but rather it performs better.

Network latency and bandwidth are important factors influencing the playability of network games. Packet drop rate is the most important factor which can cause network latency. Therefore, we investigated the effects of packet drop rate on the two P2P approaches. The experiments showed that the Tree-based P2P network system with Bucket Synchronization algorithm is suitable for Internet games, which suffer from unpredictable network latency and packet drops. We postulated that this is caused by not only the deduction of latency from the Graph to Tree conversion but also the relay mechanism of Tree-based system. Future work will investigate this issue in detail.

REFERENCES

- [1] Shuster, L., *Global Gaming Industry Now a Whopping \$35 Billion Market*, Compiler, July, 2003

- [2] Doom World, *Doomworld -- The definitive source for Doom news, information and development*, <http://doomworld.com/>, 2003
- [3] Smed, J., Kaukoranta, T. and Hakonen, H., *Aspects of networking in multiplayer computer games*. In Proceedings of International Conference on Application and Development of Computer Games in the 21st Century, HK, 2001.
- [4] Baughman, N. and Levine, B., *Cheat-proof playout for centralized and distributed online games*. In Proc. Infocom 2001, April 2001.
- [5] Ouri Wolfson, *The overhead of locking (and commit) protocols in distributed databases*, ACM Transactions on Database Systems (TODS), v.12 n.3, p.453-471, Sept. 1987
- [6] Laurent, G., and Christophe, D., *Design and evaluation of MiMaze a multi-player game on the Internet*, In Proceedings of Multimedia Computing and Systems IEEE International Conference, p233-236, 28 June-1 July 1998.
- [7] Singhal, S., *Effective remote modelling in large-scale distributed simulation and visualization environments*. PhD dissertation. Department of Computer Science, Stanford University, Palo Alto, August 1996.
- [8] Vogel, J., and Martin, M., *Network Games: Consistency control for distributed interactive media*, In Proceedings of the ninth ACM international conference on Multimedia, October 2001.
- [9] David R. Jefferson. *Virtual Time*, ACM Transactions on Programming Languages and Systems, 7(3):404–425, July 1985.
- [10] E. W. Dijkstra: *A note on two problems in connexion with graphs*. In: *Numerische Mathematik*. 1 (1959), S. 269–271
- [11] Singhal, S., and Michael, Z., *Networked Virtual Environments: Design and Implementation*, Addison Wesley, ACM Press, July 1999.
- [12] Paul, M. S., Matthew, D. R. and David, J. R., *A Local Perception Filter for Distributed Virtual Environments*, IEEE Virtual Reality Annual International Symposium (VRAIS 98), Atlanta, GA, 14-16 Mar., 1998.