Reducing Bandwidth Requirements of Consistency Maintenance Algorithms in Distributed Network Games

Kyung Seob Moon, Vallipuram Muthukkumarasamy, and Anne Thuy-Anh Nguyen
School of Information and Communication Technology, Griffith University, Australia
Gold Coast Campus, PMB 50, GCMC Queensland 9726, Australia
{k.moon, v.muthu, a.nguyen}@griffith.edu.au

Abstract

The architecture of network games is generally of two main types: Client/Server (C/S) and Peer to Peer (P2P). Distributed network games use the P2P architecture mainly to reduce network latency. This architecture may be based on graph or tree structures. Given an identical amount of data to be transferred, the bandwidth requirement in a tree-based P2P network game is not the same as that of a graph-based game. In a graph-based P2P network game, the bandwidth requirement of each player is a linear function of the number of players. In a tree-based game, by contrast, it is a quadratic function of the number of child nodes. This implies that, due to limited bandwidth, some nodes may suffer from being overwhelmed by the arrival of a large number of packets, leading to a packet-drop. In turn, the packet-drop would trigger packet retransmission; this may result in a repetitive cycle of packet retransmission and packet-drop. Such packet-drop problems may cause severe network latency. In this paper we examine methods to reduce the bandwidth requirements of tree-based P2P network games, with a view to improving system performance.

1. Introduction

In 2004 online gaming subscription revenue was estimated to be more than US $1.09 billion in the Asia/Pacific region (excluding Japan), roughly 30% more than in 2003. This figure is expected to double by 2009 [1]. The popularity of online games arises from the variety of play strategies which are not available when human users play with an AI controlled computer. For example, the shareware computer game *Doom* proved a major success on the game market at least in part because of its early support of not only the human vs. computer mode but also various human vs. human modes such as co-operative and death-match modes [2].

Among the available network architectures, the C/S architecture is preferable for Massively Multiplayer Online Role-Playing Games (MMORPG), due to ease of billing, authentification, consistency maintenance, and so on. Generic C/S architecture based games employ a tree structure where the server is placed on the root node and all clients are immediately attached to the server as child nodes. The C/S architecture aggravates network latency, due to additional data transfer caused by server-side authorizations for the commands issued from clients. Distributed network games utilize the P2P network architecture mainly to reduce network latency. Each player's states are maintained by the player and the result of the player's command or the command itself is transmitted to other players. As a trade-off, maintaining consistency is one of the most pressing problems in distributed network games. By contrast, the C/S architecture does not suffer from any inconsistency problems because the states of all players are maintained in the server only [3].

The consistency maintenance algorithms can be divided into two categories according to the methods of handling the inconsistency. The first type is called a conservative algorithm which prevents inconsistency from the beginning by making sure that the commands to proceed are safe to execute. If not, then execution is delayed until safety is assured. The second category is called optimistic algorithms whereby players' commands are processed without the safety assurance. When an inconsistency is detected, the process rolls back to the time of inconsistency to solve the problem by re-arranging and executing ordered commands in a timely fashion. Therefore optimistic algorithms perform better than conservative ones in terms of game execution speed, but the rollback process, when it becomes necessary, can cause irritation and unacceptable confusion to players. Overall, the optimistic approaches may not be suitable for network games, especially if such rollbacks are likely to be frequent or severe.

To overcome the network latency problem in the conservative approach, we proposed a tree-based P2P network system which attempts to find optimal paths for each player in the network [4]. We shall utilize packet transmission schemes which reduce network latency by increasing network bandwidth requirements [5]. But to minimize the increase in bandwidth, a packet aggregation method is proposed which exploits the advantages of tree structures as opposed to graph structures.

Two conservative algorithms are used to test the efficiency of the aggregation method in a tree structure. The first is the Lockstep algorithm, and the second is called Locked Bucket-Synchronization algorithm (LBSA) which will be explained in Section 2, together with various consistency maintenance algorithms. A theoretical analysis of tree structures and the aggregation algorithm is presented in Section 3. Experimental details and results are described in Section 4. Further analysis and discussion are presented in Section 5, and the main conclusions are given in Section 6.

2. Related work

The Lockstep algorithm [6] 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 longer than the frame interval. For example, if the game's FPS (Frames 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 [7] approach eliminates the slowdown-effect of the Lockstep algorithm by frequently transmitting the status of objects in game sessions. Generally, an unreliable protocol such as User Datagram Protocol (UDP) is used with this approach to alleviate the heavy overhead of using a reliable protocol such as Transmission Control Protocol (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 [8] and Local Lag [9] introduce artificial delays so as to synchronize a node's own frame with other nodes' frame by taking advantage of imperfect human visual perception. This approach is analogous to the buffering method of streaming audio. Even though the playout delay is extended, it still requires inconsistency resolution algorithms when network latency is longer than the extended playout delay. The playout delay is the time difference between when players' commands are generated and when they are executed and appeared on the players' screen. The approaches used in Laurent *et al* [8] may not require high bandwidth due to their adoption of multicasting in their solutions, but currently multicasting is disabled in most routes in the Internet except experimental networks such as M-Bone. While this approach uses Dead Reckoning algorithms to solve the inconsistency, the algorithm can not provide global event ordering due to its limitations.

Due to the shortcomings of consistency maintenance mechanisms in the bucket-synchronization algorithm (BSA), we proposed the LBSA [5]. The major difference between the BSA and the LBSA is the mechanism to handle inconsistency when it happens. In the case of inconsistency, the BSA simply ignores it or a convergence process begins when the threshold of state difference between players is exceeded. The LBSA adopts the method of the Lockstep algorithm which is a send-and-wait mechanism. In the LBSA, the process waits until the current frame's corresponding bucket is filled with packets of all players. When the delayed packet arrives, it is stored in the corresponding bucket and the player's game process moves forward again. To prevent a dead-lock situation [10], each player sends packets at every frame even if there is no command to transmit [5].

Dead Reckoning [11] algorithms interpolate and/or extrapolate missing and/or incoming information to reduce bandwidth requirement and latency. When the difference between actual and predicted object states exceeds a threshold, then convergence must occur. This convergence is not a global event re-ordering but simply an inaccuracies correction. The Local Perception Filter approach [12] also utilizes the limitation of human eye perception by altering the speed of objects in network games, to hide network latency. The Time Warp algorithm [13] has been introduced to solve inconsistency and/or network latency problems by adapting optimistic approaches. However, the taxing overhead of the rollover process is unavoidable.

3. Tree-based distributed network games

Distributed network games can be organized using graph or tree structures. Generally, a complete graph structure is preferable to a tree structure due to the simplicity of implementation. We assume that the graph structure is a connected graph throughout this paper unless otherwise stated. The tree structure is a converted form of graph structure by arranging each node in the view of a root node. The root node is a player and it establishes links (in other terms, edges or arcs) to other nodes directly or indirectly depending on network variables such as latency and/or bandwidth.

3.1. Graph vs. tree

On one hand, each node has direct links to other nodes and transmits its own status n-1 times (n is the number of nodes in the game sessions) in mesh structure. Therefore, assuming we must send each frame information during the game sessions, for example, we employ conservative consistency maintenance algorithms during the game sessions, the number of total packets for one frame can be obtained by the formula n(n-1).

On the other hand, there are also indirect links in the tree structure. Therefore, packets need to be relayed, which means players transmit not only their own packets but also other players' packets. The total number of packets to be transmitted for each frame is identical to each other. However, the transmitted number of packets per node is the same for each node in graph structure but it is not the same in tree structure.

3. 2. Packet relay

Packet relay does not happen in graph structure, which means each node sends its own packets only and the number of packets per frame is a linear function of the total number of nodes. Graph structure is a special case of tree structure which has no indirect links between nodes; this means that the number of non-immediate child nodes is zero and, therefore, there is no packet replay between nodes.

The total number of packets per frame can be divided into two categories, such as the number of transmitted and received packets per frame. The number of received packets per frame (NRPF) in mesh and tree structure is identical because the packets are from each node. The number of transmitted packets per frame (NTPF) is, however, different for different nodes in tree structure due to packet relay. Assuming there is no direct link between immediate child nodes in the tree structure of node A, the NTPF for node A can be calculated using Eq. (1). This equation can be expressed with the number of total nodes n and the number of immediate child nodes, yielding Eq. (2).

$$NTPF(A) = \alpha^2 + \beta(\alpha - 1) \qquad (1)$$

$$NTPF(A) = n(\alpha - 1) + 1 \qquad (2)$$

n: the total number of nodes

α : the number of immediate child nodes

β : the number of non-immediate child nodes.

As can be seen from the above equations, NTPF(A) is a quadratic function of α , the number of immediate child nodes (ICN) in the worst case where ICNs do not have direct links between them. If ICN is 1 then NTPF(A) is 1 according to the equations and it means high reduction rate of bandwidth requirement for the node. However, the node which has direct links to all other nodes should transmit packets $(n-1)^2$ times. This fact implies that bandwidth requirement of the node and overhead balancing are important factors to consider when establishing tree structure P2P systems.

3. 3. Packet aggregation

When packet-relay happens, the relay-node immediately sends the packet to reduce network latency. Also, the node needs to send its own packets to the destination. Assuming we use conservative consistency algorithms, there will be no additional delay if the node waits for relay-packets and aggregate all packets including its own packets for sending to each destination. This approach can reduce bandwidth requirement of the node by decreasing NTPF of the node. When a packet is sent, it is transmitted with additional information which is called packet header. This packet aggregation method can reduce the number of packet headers, therefore it reduces the bandwidth requirement of the node.

The best case of packet aggregation is that a shortest path exists which goes through all nodes. For example, if there are 3 nodes then A-B-C route is shorter than A-C. In this case, node B will be a super node that connects all other nodes. In the best case, the number of total packets per frame in tree structure with aggregation method can be obtained using the expression 2(n-1) where n is the number of nodes. In the worst case, it will be *n(n-1)* and this is the case of mesh structure and there is no packet aggregation at all. We can write an expression to indicate the average value by adding the above two expressions and dividing them by 2. Eq. (3) shows the detail.

$$(n(n-1) + 2(n-1)) / 2 = ((n-1)(n+2))/2 \qquad (3)$$

The number of packets per frame in tree structure is always smaller than in mesh structure when n is greater than 3, as can be seen in Eq. 3. (We omit the proof due to space limitations.) Generally, P2P requires more than three nodes. Therefore, we can guarantee that the packet aggregation method reduces bandwidth requirement in tree structure relative to relay-only method in any case. The number of packets per frame is a linear function of the number of nodes in the best case and it is a quadratic function in the worst case.

3. 4. Retransmission

Retransmission is unavoidable when packet drop or errors occur, especially when frequent packet regeneration scheme is not used. The decision for retransmissions is based on acknowledgement timeout and acknowledgement 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.

3. 5. Acknowledgement

Generally, network latency between players in the Internet is neither symmetric nor fixed. However, for clarity of efficiency analysis of consistency maintenance algorithms, we assume that network latency is fixed in the network simulator. Therefore, acknowledgement time calculation is based on RTT (round-trip time) measurement. In real life situations, RTT between players is not constant during game sessions in the Internet but in this experiment we just use the exact time of RTT for clearer analysis of the algorithm efficiency. Each player sends frame packets at regular intervals but acknowledgement packets are sent immediately when other players' packets arrive. This tree based P2P system can prevent acknowledgement implosion problem [3] because each node manages its own child or children.

4. Experimental results

4.1. Experimental data set

In general, most P2P network-based games support eight players due to network constraints such as latency and bandwidth. Therefore, we created the simulator initially with a generated data set of eight players, and randomly selected network latency with values between 5 and 100 ms. Table 1 shows maximum and average latency between each node. Maximum latency, a key factor which affects the overall game speed, for the eight -player experimental data set is 77. We will expand further on the meaning of maximum latency in the next section.

Table 1: Latency values between eight players (Max Latency: 77, Average Latency: 40.46)

	0	1	2	3	4	5	6	7
Max:	76	64	77	76	77	77	77	70
Avg.:	41	43	44.43	31.57	34.57	44	46.57	38.57

4.2. Simulator architecture

Our 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, Priority Queue for a re-sending buffer, and Circular Array for a game buffer, in order to store other players' packets, to re-send dropped packets, and to gather frame data respectively. 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 analysis. The Player class is in charge of packet relay, sending acknowledgement, frame packet generation and resending when packet drop is detected [3].

4. 3. Comparison between algorithms

Two consistency maintenance algorithms, Lockstep (LS) and Locked Bucket-Synchronization algorithm (LBS), are implemented for this experiment. The experiment duration is 60 seconds and frame interval is 100 ms because the maximum network latency is set as 100 ms between players. Playout delay for LBS is 200 ms and two network structures are used, namely graph and tree.

Table 2: Experimental results of Lockstep algorithm on graph structure

Drop Rate (%)	Total Packets	FPS	Avg. Packets per Frame
0	67066	10.00	111.78
1	51612	7.58	113.43
2	47517	6.88	115.05
5	39398	5.47	120.12
10	35410	4.52	130.66

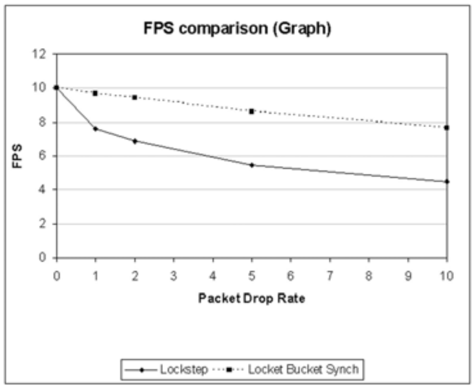

Figure 1: The changes of FPS according to packet drop rate on graph structure

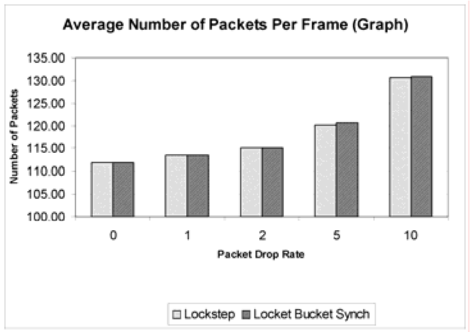

Figure 2: The changes of average number of packets per frame according to packet drop rate on graph structure

Table 2, Figure 1 and 2 show experimental results based on LS and LBS algorithms under graph structure. The first row in the table shows the optimal results of the game session which has no packet drop at all. Therefore, FPS (Frame per Second) value is 10 and this value is same as the optimal value. In optimal situation, the formula for the total number of packets per frame is expressed as $n(n-1)\times 2$ where n is the number of players. A factor of 2 is included because we also count acknowledgement packets even though their size is smaller than the size of frame packets. Therefore, the value for the Average Packets per Frame for eight-player game sessions is 112 as shown in the table. Figure 3 and 4 display the results of experiments using the two algorithms under tree structure.

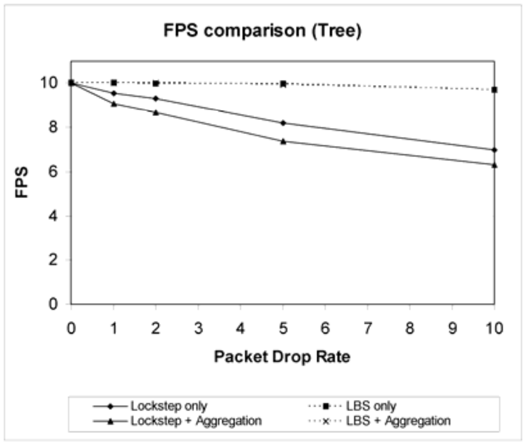

Figure 3: The changes of FPS according to packet drop rate on tree structure

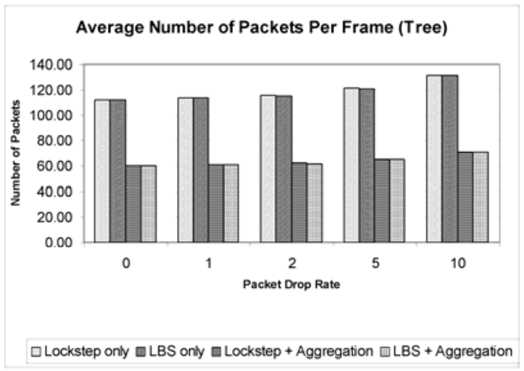

Figure 4: The changes of average number of packets per frame according to packet drop rate on tree structure

5. Analysis and discussion

As shown in Figure 1, LBS algorithm performs better than LS on packet drop rate, in terms of game execution speed measured by FPS. LS algorithm achieved around 45% of optimal value and LBS algorithm achieved about 76% on 10% packet drop rate. The average numbers of packets per frame under graph structure are almost identical for both algorithms as displayed in Figure 2. This implies that LBS algorithm performs better than LS algorithm in terms of game execution speed without increasing bandwidth requirement for players.

To reduce network latency and bandwidth requirement, tree-based P2P system is introduced and the experimental results are shown in Figure 3 and 4. The FPS values are almost optimal (about 97%) when LBS algorithm is employed on tree structure even with 10% packet drop rate as shown in Figure 3. The numbers of packets per frame for each algorithm on tree structure are almost identical. However, when aggregation method is utilized, the number of packets per frame is dramatically reduced from 112 to 60 on 0% of packet drop rate. The number of packets per frame for eight-player game sessions is 63 when packet aggregation is available in average case according to Eq. (3). The number becomes 112 in the worst case as clearly shown in Figure 4.

6. Conclusions

This research we proposed a packet aggregation method for tree structure in P2P multiplayer distributed network games with the aim of reducing network latency and bandwidth requirements. The tree structure finds the shortest path for each player, in order to reduce network latency, and the packet aggregation method waits until all packets bound for one destination arrive, then aggregates the packets and transmits the aggregated packet.

We also examined the effects of the aggregation method on two conservative consistency maintenance algorithms, Lockstep and Locked Bucket Synchronization. To compare the efficiency of the aggregation method, experiments were conducted using the network simulator described in Section 3, with the network setting of 8 player nodes.

The LBS algorithm with aggregation method performs better than any other combination in terms of game frame rate and bandwidth requirement. However, LBS algorithm prolongs playout-delay which also affects game playability and this may be critical to some genres of network games. Therefore, in continuing research, we will further examine the effects of the playout-delay on distributed network games. Also, to analyze the efficiency of the packet aggregation method, we will apply the method to game sessions of varying numbers of players.

References

[1] Heng, S. Y., *Online Gaming Subscription Revenue Surpassed US$1 Billion in 2004 and Will More Than Double by 2009*. IDC – Press Release, Available: http://www.idc.com/getdoc.jsp?containerId=pr2005_08_04_110417 [04 Aug 2005], 2005

[2] Doom World, *Doomworld -- The definitive source for Doom news, information and development*. Available: http://doomworld.com/ [15 December 2003], 2003

[3] Moon, K. S., Muthukkumarasamy, V., Nguyen A. T., and Kim, H. S., Maintaining Consistency in Distributed Network Games. *In Proceedings of the 13th IEEE international conference on networks jointly held with the 7th IEEE Malaysia international conference on communications*, Kuala Lumpur, Malaysia, pp. 374-379, 2005

[4] Moon, K. S., Muthukkumarasamy, V., and Nguyen A. T., Efficiently Maintaining Consistency Using Tree-Based P2P Network System in Distributed Network Games. *The Edutainment 2006, International Conference on E-learning and Games*, Hangzhou, China, 2006

[5] Moon, K. S., Muthukkumarasamy, V., and Nguyen A. T., Reducing Bandwidth Requirements on Consistency Maintenance Algorithms in Distributed Network Games. *IADIS International Conference, Applied Computing 2006*, San Sebastian, Spain, 2006

[6] Baughman, N. E. and Levine, B. N., Cheat-proof playout for centralized and distributed online games. *In Proceedings of IEEE Infocom*, Anchorage Alaska, USA, vol. 1, pp. 22-26, 2001

[7] Singhal, S., and Zyda, M., Networked Virtual Environments: Design and Implementation, Addison Wesley, ACM Press, July 1999.

[8] Laurent, G., and Diot, C., 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.

[9] Vogel, J., and Mauve, M., Network Games: Consistency control for distributed interactive media, *In Proceedings of the ninth ACM international conference on Multimedia*, October 2001.

[10] Wolfson, O., 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

[11] 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.

[12] Sharkey, P. M., Ryan, M. D. and Roberts, D. J., A Local Perception Filter for Distributed Virtual Environments, *IEEE Virtual Reality Annual International Symposium (VRAIS 98)*, Atlanta, GA, 14-16 Mar., 1998.

[13] Jefferson, D. R., Virtual Time, *ACM Transactions on Programming Languages and Systems*, 7(3):404-425, July 1985.