

# REDUCING NETWORK LATENCY ON CONSISTENCY MAINTENANCE ALGORITHMS IN DISTRIBUTED NETWORK GAMES

Kyung Seob Moon, Vallipuram Muthukkumarasamy, 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](mailto:{k.moon, v.muthu, a.nguyen}@griffith.edu.au)

## ABSTRACT

Network latency and bandwidth are important factors in online games for satisfying users' playability. On one hand, the development of networking technology enables the spread of broadband Internet connection, which is now common in developed countries. Therefore, the bandwidth problem is less significant than the network latency problems. On the other hand, due to the physical properties of data transmission it is hard to solve the network latency problems. Consistency is another important factor in online games for state update and player action fairness. Event ordering is one of the methods for maintaining consistency in traditional multi-process environments. The network latency and consistency maintenance are mutually exclusive relationship. Therefore, we will examine various consistency maintenance algorithms and suggest transmission schemes which can reduce network latency without increasing bandwidth requirement significantly. Our experimental results also show the effects of transmission schemes with consistency maintenance algorithms on game sessions of various numbers of players.

## KEYWORDS

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

## 1. INTRODUCTION

The online gaming subscription revenue was more than US \$1.09 billion in Asia/Pacific regions excluding Japan in 2004. It is roughly 30% more than in 2003 and expected to be doubled by 2009 (Heng, 2005). The popularity of the online games comes from the variety of play strategies which are not available when human users play with AI controlled computer. Shareware computer game Doom hit a great success on the game market and it supports not only human vs. computer mode but also various human vs. human modes such as co-operative and death-match modes (Doom World, 2003). To maintain consistency of game states between players, this game uses a frequent transmission scheme which sends player state packets frequently and regularly (Singhal et al., 1999). This routine is performed even when there is no change in states, wasting network bandwidth and restricting the maximum number of players in network games.

Network latency is another important factor in achieving playability in network games. On one hand, because of the widespread of broadband internet access, the network bandwidth problem is not as serious as in the past. On the other hand, network latency problem is hard to solve due to the physical characteristics related to data transmission. The latency in typical Ethernet LAN is generally less than 0.3ms and varied in WAN environments (Cheshire, 2003). The main cause of network latency is the distance between players, but network situation such as congestion can be a major contributing factor. According to Stuart's experiments (Cheshire, 2003) it took 84.5ms for a round-trip from the east coast to the west coast of the United States. Usually, round-trip time (RTT) for international transmission is longer. According to the preliminary experiments we performed, it took about 300ms for a round-trip from the Gold Coast in Australia to Seoul in Korea. We will examine methods that improve network latency by sacrificing network bandwidth in Section 3.

Another factor that will affect playability is the game architecture. Online game architectures can be divided into two categories which are Client/Server (C/S) and Peer-to-Peer (P2P). The centralized game architectures are C/S based and widely used for most of MMORPG games due to the simplicity of consistency maintenance, better security, improved authentication, and easy billing system. However, this architecture introduces additional network latency compared to distributed game architectures and this can cause a network bottle neck when lack of network resources happens. Distributed game architectures can eliminate redundant packet transmissions between server and client to reduce network latency. Because each node manages its own object states it suffers from the hardship of maintaining consistency of game states between players due to network delay (Jiang, 2005).

The consistency maintenance algorithms can be divided into two categories according to 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 will wait until safety is assured. The second category is called optimistic algorithms whereby players' commands are processed without the safety assurance. Whenever an inconsistency is detected the process rollbacks and tries to solve the problem. Therefore optimistic algorithms perform better than the conservative type in terms of game execution speed but the rollback process can cause irritation and confusion to players. Overall, the optimistic approaches may not be suitable for network games. To overcome the network latency problem in the conservative approach, we propose packet transmission schemes which reduce network latency by increasing network bandwidth requirement.

Two conservative algorithms are used to test the efficiency of transmission schemes. The first is the lockstep algorithm which will be explained in Section 2 with various consistency maintenance algorithms. The second algorithm is called locked bucket-synchronization, explained in Section 3 in detail together with the structure of the network simulator which we implemented for the experiments. Experimental details and results are described in Section 4. Analysis and discussion are presented in Section 5. Conclusions are given in Section 6.

## **2. RELATED WORK**

The Lockstep algorithm (Baughman et al., 2001) is one of the simplest solutions for consistency maintenance in the P2P structure. Each peer waits for other peers' packets of 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 (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.

Frequent State Regeneration (Singhal, 1999) approach eliminates the slowdown-time of the lockstep 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 heavy 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 (Laurent et al, 1998) and Local Lag (Mauve et al., 2004) introduce artificial delays so as to synchronize a node's own frame with other nodes' frame by utilizing imperfect human visual perception ability. This approach is analogous to the buffering method of streaming audio. Even though the playout time is extended, it still requires inconsistency resolution algorithms when network latency is longer than the extended playout time. The approaches used in Laurent et al 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.

Dead Reckoning (Singhal, 1996) algorithms interpolate and/or extrapolate missing and/or incoming information to reduce bandwidth requirement and latency. When the difference between actual object states and predicted ones exceed a threshold then convergence happens. This convergence is not the global event re-ordering but simply an inaccuracies correction. The Local Perception Filter approach (Sharkey et al., 1998) also utilizes the limitation of human eye perception by altering the speed of objects in networking games for

hiding network latency. The Time Warp algorithm (Jefferson, 1985) 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. LOCKED BUCKET-SYNCHRONIZATION ALGORITHM & TRANSMISSION SCHEMES

#### 3.1 Locked Bucket-Synchronization Algorithm

One of the main problems of the Lockstep algorithm is the irregular playout delays due to network latency. The playout delay is the time difference between when players' commands are generated and when they are executed and appeared on the players' screen. Responsiveness or response time is an alternative term for playout delay (Mauve, 2004). As described in Section 2, to prevent irregular playout delays, the Bucket-Synchronization algorithm extends the playout delay. It is analogous to the buffering mechanism of streaming audio and video. As can be seen in Figure 1, commands issued between Frame 0 and Frame 1 are stored in Bucket 0 and executed at Frame 2. The major difference between the bucket-synchronization algorithm (BSA) and the locked bucket-synchronization algorithm (LBSA) is the mechanism to handle inconsistency when it happens. In that case, the former simply ignores it or convergence process begins when the threshold of state difference between players is exceeded. The latter adopts the method of the Lockstep algorithm which is send-and-wait mechanism. In the LBSA, the process waits until 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 (Wolfson, 1987), each player sends packets at every frame even if there is no command to transmit. In Figure 1, the empty circle represents packets without commands.

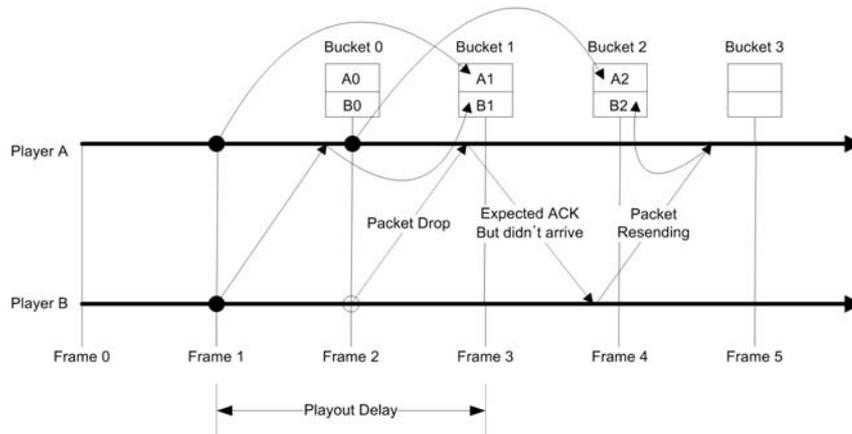


Figure 1. The Locked Bucket-Synchronization algorithm.

Similar algorithm is suggested in Baughman et al, 2001. However, the algorithm is used for network game security and introduces additional network latency or processor cycles. The LBSA commits its commands in corresponding buckets without using the two-phase commitment described in Baughman et al. because it assumes the lookahead cheat can be detected using RTT (Round-trip time) measurement methods. For example, we assume that player A's IP is 132.234.xxx.5 and player B, 61.41.yyy.3. Each player measures RTT to the opponent's IP address and neighboring IP addresses before game session starts. When the opponent reports unusual network latency then the player may be performing the lookahead cheat. Then by agreement between other players, the cheat player will be banned from the game session. Discussion of the agreement scheme is omitted here due to space limitation.

## 3.2 Transmission Schemes (Blind and Smart Transmission Schemes)

We suggest two transmission schemes, blind and smart, in this paper. As the name implies, the former always sends the same packets twice. Therefore, bandwidth requirement for each player will be nearly doubled, compared to the no transmission scheme. Lincroft (1999) used the aggregated packets transmission scheme in his “X-Wing vs. TIE Fighter” game that aggregates previous frame and current frame packets together before transmitting them. This scheme also doubles bandwidth requirements similar to the blind transmission scheme (BTS). Doubling bandwidth requirement is not a favoured feature in designing multiplayer games because it will constrain game design significantly.

The smart transmission scheme (STS) checks the future arrival time of packets to determine whether the packet being sent is critical or not. The meaning of criticality here is the possibility of the game being delayed if the packet is dropped. This can be determined by comparing two values, must-arrive time and next packet arrival time. The must-arrive time (MAT) is the time when the packet must arrive so as not to delay whole game process. It depends on the playout time of the game. In the Lockstep algorithm, packets must arrive before the next frame starts. The MAT is the addition of packet sending time and the playout delay in the bucket-synchronization algorithm. The next packet arrival time (NPAT) is the time when the next packet arrives if the packet drop is detected and retransmitted. If the packet being sent is determined as a critical packet then it is sent twice one after another.

## 3.3 Packet Transmission

TCP is a reliable protocol which means that it assures packet arrival and packet ordering. However, the overhead of the protocol is burdensome compared to other unreliable protocols such as UDP (Bettner et al., 2001). Also, the locked bucket-synchronization algorithm uses frequent state regeneration scheme so it is important not to waste bandwidth by using low overhead protocol such as UDP. As this protocol does not guarantee packet arrival, we need to ensure the arrival of packets and when packet drop happens then transmission needs to occur. The packet drop can be detected using packet acknowledgement: when packet arrives the recipient sends back an acknowledgement. In the implementation of the network simulator explained in Section 3.5, we use a timeout scheme for packet retransmission which means packet retransmission occurs when the acknowledgement does not arrive in time.

## 3.4 Acknowledgement

Generally, network latency between players in the Internet is neither symmetric nor fixed. However, for clarity of efficiency analysis of transmission and 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 situation, 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.

## 3.5 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 (Moon et al, 2005).

## 4. EVALUATION OF EXPERIMENTAL RESULTS

### 4.1 Experimental Data Set

In general, most of P2P network-based games support between two and sixteen players due to network constraints such as latency and bandwidth. Therefore, we created the simulator initially with generated data sets of four, eight and sixteen players, and randomly selected network latency with values between 5 and 100 ms. Table 1, 2 and 3 show maximum and average latency between each node. Travel distances between nodes in milliseconds are also presented in Table 1, but not in Table 2 and 3 due to space limitation. Maximum latency, a key factor which affects the overall game speed, for the four, eight, and sixteen-player experimental data set are 82, 77, and 99 respectively. We will explicate the meaning of maximum latency further in the next section.

Table 1. Latency values between four players (Max Latency: 82, Average Latency: 46.83)

	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
Average:	41	43	44.43	31.57

Table 2. 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
Average:	41	43	44.43	31.57	34.57	44	46.57	38.57

Table 3. Max and average latency values between sixteen players (Max Latency: 99, Average Latency: 52.10)

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

### 4.1 Comparison between algorithms

Two consistency maintenance algorithms, Lockstep and Locked Bucket-Synchronization algorithm (LBS), are implemented with two transmission schemes, blind and smart (BTS and STS) 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. Payout delay for LBS is 200 ms and three player groups are used which are 4, 8 and 16 player groups.

Table 4, Figure 3 and 4 show experimental results which are based on Lockstep algorithm and two types of transmission schemes. 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 column in Table 4 should be 24. However, according to the Table, the value is 23.95 because the experiment only counted packets which arrived in 60 seconds sharp.

Drop Rate (%)	Total Packets	Dropped Packets	FPS	Avg. Packets per Frame
0	14368	0	10.00	23.95
1	13679	117	9.40	24.25
2	12755	233	8.63	24.62
5	11317	597	7.25	26.02
10	10134	998	6.05	27.92

When the Blind Transmission Scheme (BTS) is applied, the number of packets transmitted doubled. It can be observed clearly in Figure 4. However, the FPS values are increased positively. With 10% of packet drop rate in lockstep and no transmission scheme combination in Table 4, the FPS dropped down to 6.05 which is 60% of full game execution speed. However, with BTS and 10% packet drop rate, the FPS became 9.02 in Figure 3. Furthermore, Figure 3 and 4 show that the BTS performs well without variation compare to the no transmission scheme with lockstep algorithm.

Also the experimental result with Lockstep algorithm and Smart Transmission Scheme (STS) is shown in Figure 3 and 4. According the result, STS reduces the total number of packets but the FPS values are almost identical to those of BTS. However, when the packet drop rate increases, the total number of packets per frame increases and becomes very similar to the corresponding one in BTS. For example, STS with Lockstep sends about 40 packets per frame when the packet drop rate is 10%, which is not so different from BTS with Lockstep that is 46 packets per frame.

Figure 3 and 4 show the changes of FPS and the average number of packets per frame in four-player-game sessions. As can be seen, the proposed Locked Bucket-Synchronization (LBS) algorithm performs better than the Lockstep algorithm in terms of game execution speed (FPS) and bandwidth requirement. With BTS and LBS combination, the total number of packets is almost identical to the BTS and lockstep algorithm combination. With STS and LBS, the total number of packets becomes only about 10% more than the LBS without transmission scheme, but the FPS value is almost optimal which is above 92% of full game speed in any drop rate cases. Furthermore, Figure 3 (a) clearly shows the effect of packet drops compared to Figure 3(b) which utilized LBS algorithm. According to the figures, FPS values are drastically decreased as the packet drop rates goes up with lockstep algorithms without transmission scheme. However, LBS algorithm performs well even without transmission schemes.

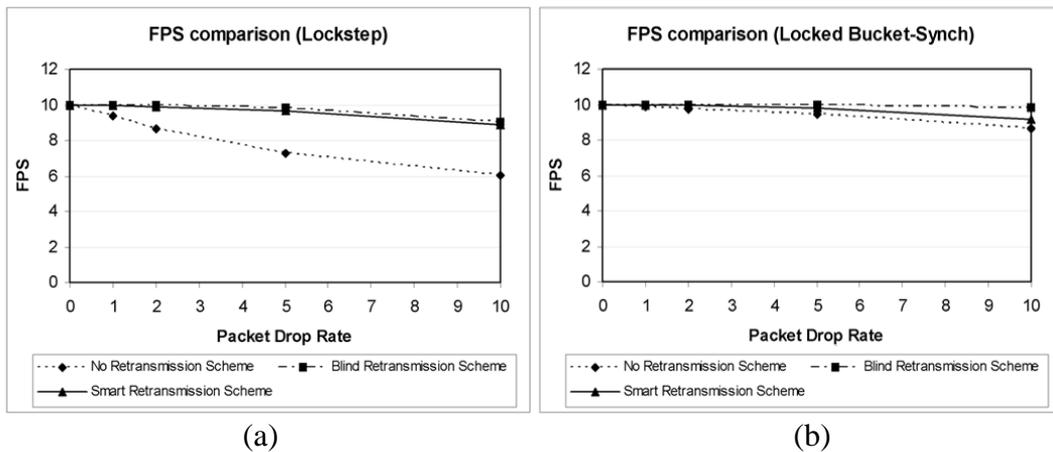


Figure 3. Four-player-game sessions: The changes of FPS according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

Figure 5 and 6 shows the changes of FPS in eight and sixteen-player-game sessions. The graphs of the changes of bandwidth requirement are omitted due to similarity of their shapes compared to Figure 4. As can be observed, the more players and higher drop rate, the poorer the performance (FPS) is without transmission schemes. This situation gets worse when the Lockstep algorithm is used. In Lockstep and transmission schemes with eight player game, about 75% of full game execution speed is achieved, but only 50% in 16 player game sessions with Lockstep and any transmission schemes combinations. This clearly shows that not

only bandwidth requirement is critical in supporting more players in network games, but also network latency is when conservative algorithms are used for consistency maintenance. To mask network latency, we propose the LBS algorithm and Figure 6 shows that it performs better than the Lockstep algorithm, in that it executes game sessions at about 80% of the full game speed even with 10% packet drop rates in sixteen-player game sessions when transmission schemes are applied.

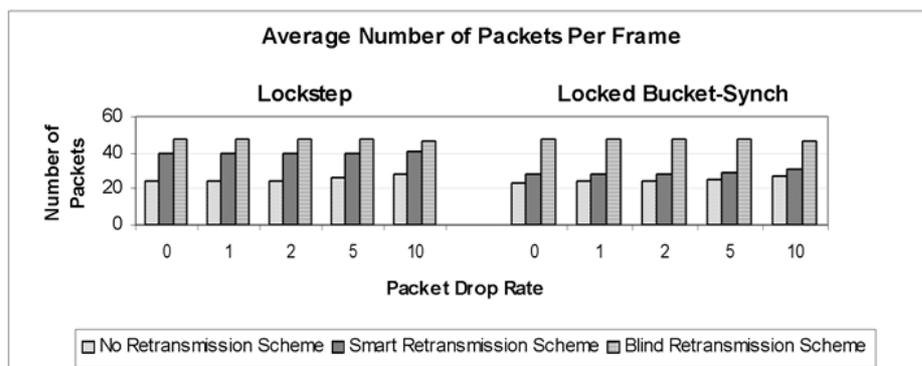


Figure 4. Four-player-game sessions: The changes of average number of packets per frame according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

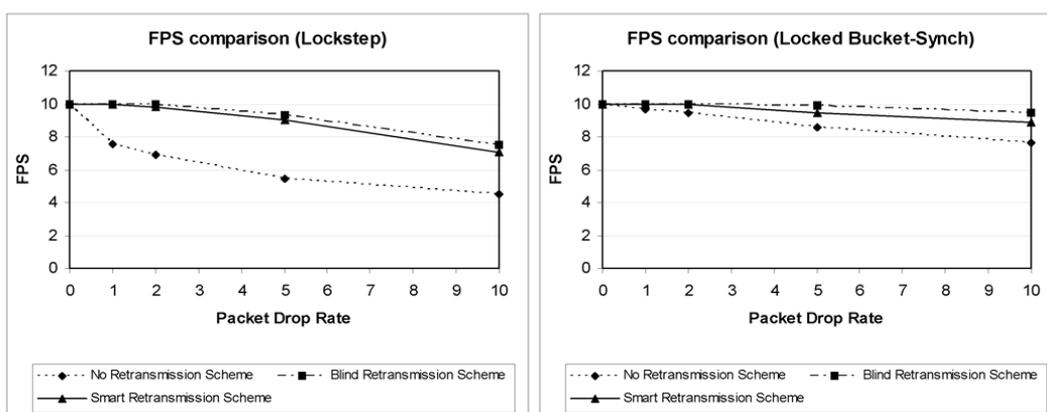


Figure 5. Eight-player-game sessions: The changes of FPS according to packet drop rate on Lockstep and LBS algorithms

## 5. CONCLUSION

In this research the authors proposed a Locked Bucket Synchronization algorithm for maintaining consistency in P2P multiplayer distributed network games. The proposed technique masks network delay utilizing human perception delay. We also examined the effects of the blind and smart transmission schemes on two conservative consistency maintenance algorithms, Lockstep and Locked Bucket Synchronization. To compare the efficiency of two transmission schemes, experiments were conducted using the network simulator explained in Section 3, with the network settings of 4, 8, and 16 player nodes.

The BTS with locked bucket-synchronization algorithm performs better than any other combinations in terms of game frame rate. However, it increases bandwidth requirement which is critical to some genre of multiplayer network games. Therefore, STS with locked bucket-synchronization algorithm is proposed and this combination performs almost identical to the BTS in terms of FPS but with significantly reduced bandwidth requirement. We will further examine the effects of the two transmission schemes on other systems such as the tree-based peer-to-peer approaches proposed in our previous works (Moon et al., 2005).

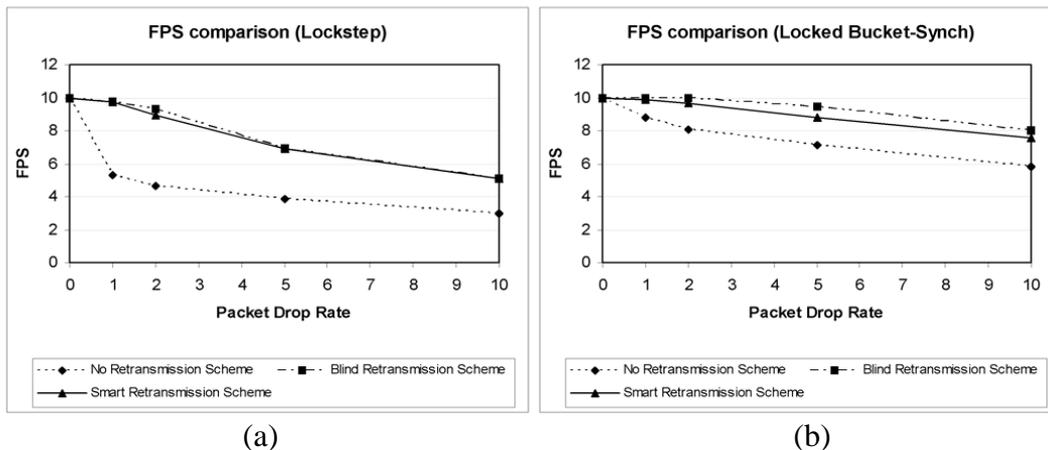


Figure 6. Sixteen-player-game sessions: The change of FPS according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

## REFERENCES

- Baughman, N. E. and Levine, B. N., 2001. Cheat-proof payout for centralized and distributed online games. *In Proceedings of IEEE Infocom*, Anchorage Alaska, USA, vol. 1, pp. 22-26
- Bettner, P. and Terrano, M., 2001. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. *Gamasutra magazine*, March 2001. Available: [http://www.gamasutra.com/features/20010322/terrano\\_01.htm](http://www.gamasutra.com/features/20010322/terrano_01.htm) [20 September 2004]
- Cheshire, S., 2003. *It's the latency, stupid*, [Online]. Available: <http://rescomp.stanford.edu/~cheshire/rants/Latency.html> [12 October 2004]
- Doom World, 2003. *Doomworld -- The definitive source for Doom news, information and development*. [Online]. Available: <http://doomworld.com/> [15 December 2003]
- Heng, S. Y., 2005. *Online Gaming Subscription Revenue Surpassed US\$1 Billion in 2004 and Will More Than Double by 2009*. IDC
- Jefferson, D. R. J., 1985. Virtual Time, *ACM Transactions on Programming Languages and Systems*, 7(3):404-425.
- Jiang, X., Safaei, F. and Boustead, P., 2005. Latency and scalability: a survey of issues and techniques for supporting networked games. *In Proceedings of the 13<sup>th</sup> IEEE international conference on networks jointly held with the 7<sup>th</sup> IEEE Malaysia international conference on communications*, Kuala Lumpur, Malaysia, pp. 150-155.
- Laurent, G. and Diot, C., 1998. Design and evaluation of MiMaze a multi-player game on the Internet. *In Proceedings of Multimedia Computing and Systems IEEE International Conference*, Austin Texas, USA, pp. 233-236.
- Lincroft, P., 1999. The Internet sucks: Or, what I learned coding X-Wing vs. TIE Fighter. *In Proceedings of Game Developers Conference*.
- Mauve, M. et al, 2004. Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Transaction on Multimedia*, vol. 6, pp. 47-57.
- Moon, K. S. et al, 2005. Maintaining Consistency in Distributed Network Games. *In Proceedings of the 13<sup>th</sup> IEEE international conference on networks jointly held with the 7<sup>th</sup> IEEE Malaysia international conference on communications*, Kuala Lumpur, Malaysia, pp. 374-379.
- Sharkey, P. M., Ryan, M. D. and Roberts, D. J., 1998. A Local Perception Filter for Distributed Virtual Environments. *IEEE Virtual Reality Annual International Symposium (VRAIS 98)*, Atlanta, GA, pp. 14-16.
- Singhal, S. et al, 1999. *Networked Virtual Environments: Design and Implementation*. Addison Wesley ACM Press, New York, USA.
- Singhal, S., 1996. *Effective remote modelling in large-scale distributed simulation and visualization environments*. PhD dissertation. Department of Computer Science, Stanford University, Palo Alto.