Cheat-Proof Playout for Centralized and Distributed Online Games

52031750 박희찬

Introduction

1. 온라인게임상에서의 부정행위는 실시간 인터렉션에서의 적절한 Playout, 큰 규모의 게임 및 통신의 확장성, 부정행위자의 발견과 방지와 근접한 관계가 있다.

2. 최근 온라인 게임에서 가장 인기있게 쓰이는 Centralized(Server-Client) 방식은 하나의 동일한 Viewport를 제공하지만 온라인상의 커다란 Virtual World에서 User의 수와 처리할 신호(통신)가 증가함에 따라 병목현상을 일으킨다.

3. 이러한 문제로 Distributed(Peer-to-Peer)방식으로 전환은 수행률과 확장성의 증가를 가져오지만 Server-Client방식에서의 Cheating문제는 이미 존재할 가능성이 있고 Player간의 인터렉션이 복잡해진다.

4. 현재 많은 온라인게임에서 cheating방지를 위한 보안은 없거나 미비한 상태이다. 기존에 흔히 사용하던 동기화 기술을 확장하여 확장성이 있고 수행률이 더 좋은 가운데 보안과 유저간의 동등함을 유지하는 방법에 대한 새로운 기술을 소개하는 것이 논문의 목표이다.

Assumptions And Terminology

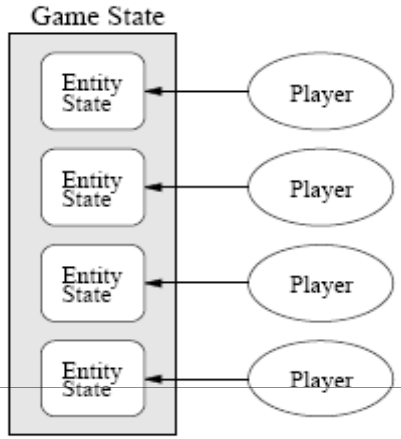


Fig. 1. Game state partitioning

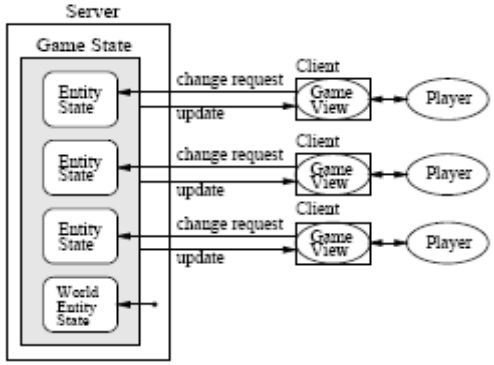


Fig. 2. Centralized-control client-server

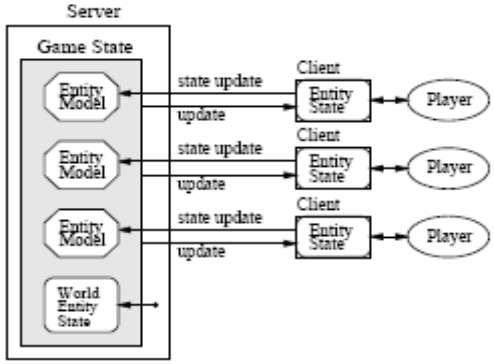


Fig. 3. Decentralized-control client-server

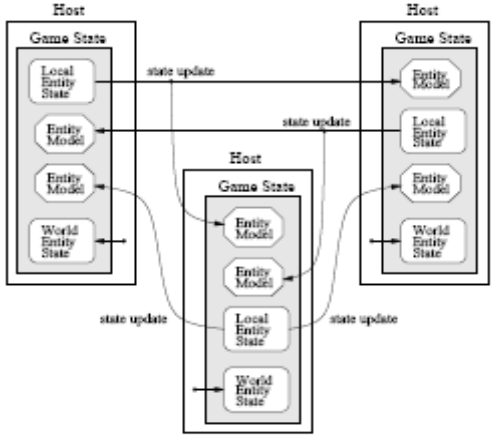


Fig. 4. Distributed

Assumptions And Terminology

1. Centralized-control
2. Decentralized-control
3. Distributed
4. Dead Reckoning
5. Asynchronous

Fair Playout

1. Irresolvable Interactions Problem

- Dead Reckoning의 사용이나 Server의 부정확한 결정, 각각의 Distributed Host의 정확하지 못할 가능성으로 발생한다.

2. Cheating Under Dead Reckoning

- 가장 느린 Player S는 가장 빠른 Player A의 속도를 따라가기 위해 Bucket을 버리게 되고 A는 S의 현재위치를 예측하게 되지만 실제 S의 위치는 알수 없게 된다. S는 n번째 Bucket에서 그럴듯한 Bucket을 만들어 전송하게 되는데 A는 이것이 cheating인지 아닌지 구분할 수 없게 된다.

Cheating-Proof Game Interaction

1. Stop And Wait 프로토콜을 사용하면 한 호스트의 인터렉션에 대해 다른 모든 호스트가 받을때까지 현재상태의 유지를 위하여 Stop and Wait상태가 된다. 이러한 특성상 Suppress-Correct Cheat은 완전히 제거된다.

2. 이러한 Stop And Wait 프로토콜을 사용하여 Suppress-Correct Cheat을 제거하였지만, 모든 호스트를 기다리는 틈을 노려 Cheating을 하게 되는데 이를 Lookahead Cheat이라고 한다. Waiting상태에서 인간의 반응속도로는 해결할 수 없는 동작를 Cheating agent를 사용하여 대응하는 문제가 생기게 된다.

Lockstep Protocol

1. Lookahead Cheat를 막기위해 Stop-and-Wait 프로토콜의 보완된 버전인 Lockstep 프로토콜을 사용한다.

2. 각 플레이어는 다음 턴에 대해 그냥 알리지 않고, 해쉬를 사용하거나 해쉬의 중복이나 충돌의 위험이 있다면 패킷내에 랜덤한 패딩을 추가하여 알린다. 이러한 평문 해쉬의 사용으로 Lookahead cheating을 막을수 있다.

3. 하지만 Lockstep protocol은 네트워크 상태의 영향으로 지연되어 전체적인 게임의 속도를 저하시키는 단점을 가지고 있다.

Proof of Correctness

1. Theorem : Lockstep protocol is safe.
   Lockstep protocol is live.

2. Proof
   - safe : Hash의 사용으로 매 프레임으로부터 나오는 이벤트에 대한 변경이 불가능하고 다음 프레임의 이벤트에 대한 예측자체가 불가능해진다.
   - live : 모든 플레이어는 신뢰적인 곳에서 통신하기때문에 모두 제한된 시간만큼 기다리게 되므로 일관적인 형태를 가지게 된다.

Asynchronous Synchronization

1. 비동기식 동기화는 각 호스트에서 다른 호스트로의 시간을 비동기식으로 진전시키지만 인터렉션의 요청되면 Lockstep-style로 동작한다. 이것은 정확하고 공정함을 보장하게된다.

2. 인터랙션에 대한 해결의 위해 시간을 많이 소비하는 플레이어와의 약해진 연결에 대한 일시적인 효과를 제거해주는 역할을 한다.

3. Lockstep mechanism의 비동기 연산은 다른 모든 플레이어와 관계없이 게임이 진행되기 때문에 수행률에 이득을 준다.

Spheres of Influence

1. 다음 턴에 플레이어의 게임영역에 침범하게 되어 영향을 줄수 있게 되는 sphere의 영역.

2. 두명의 플레이어가 다음턴에 서로의 영역을 침범하지 않는다면 결정적인 이벤트에 대해 서로 영향을 미치지 않게된다.

3. E.g.) 플레이어가 숲속의 소리가 닿는 영역에 있지 않다면 떨어지는 나무가 만들어내는 소리에 대해서 고려할 필요가 없다.

Spheres of Influence

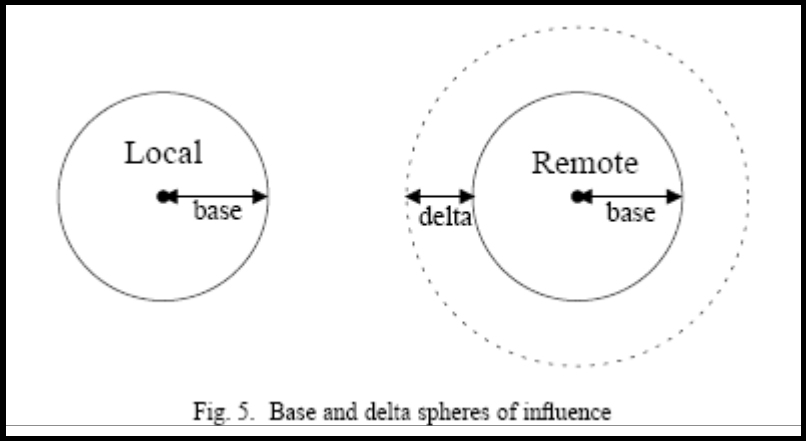


Fig. 5. Base and delta spheres of influence

l	Local host
R	The set of all remote hosts
r	A remote host in R
t	The current frame at the local host
S_t^h	State of host h at frame t
$H(S_t^h)$	Hash of state S_t^h for host h for frame t
p_t^h	Potential influence of host h at frame t

Fig. 6. Table of variables.

1. Compute S_t^l
2. Send $H(S_t^l)$
3. Process accepted $H(S_y^r)$ messages that have arrived
4. *foreach* $r \in R$
   Take next S_y^r if any have arrived where $y \leq t$
   Let frame of latest state taken be x
   compute p_t^l, and p_t^r dilated from x
   if $(p_t^l \cap p_t^r = \emptyset)$
   then record l is *not waiting* for r
   else if $H(S_t^r)$ accepted
   then l is *not waiting* for r
   else l is *waiting* for r
5. *if* not *waiting* for any r
   then send S_t^l
   resolve any interactions
   finalize and render turn t
   advance to turn $(t+1)$

Fig. 7. AS at the local player for each game turn.

Spheres of Influence

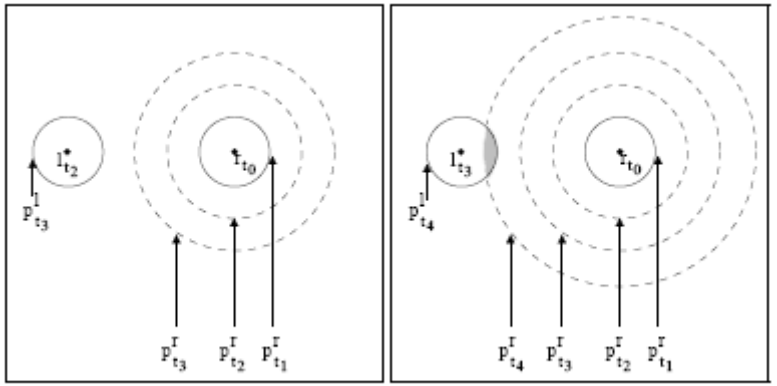


Fig. 8. (Left) Dilation to t_3. (Right) Dilation to and intersection at t_4.

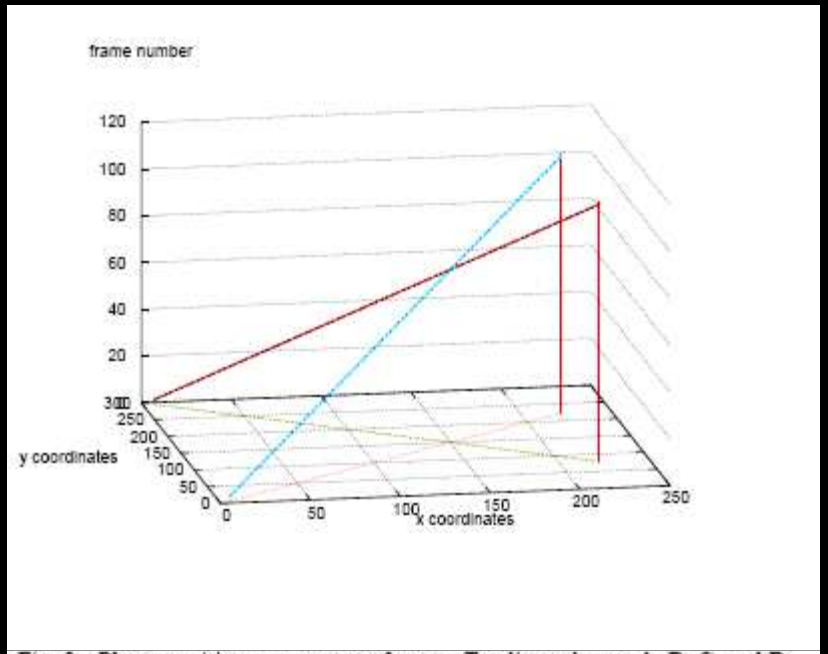
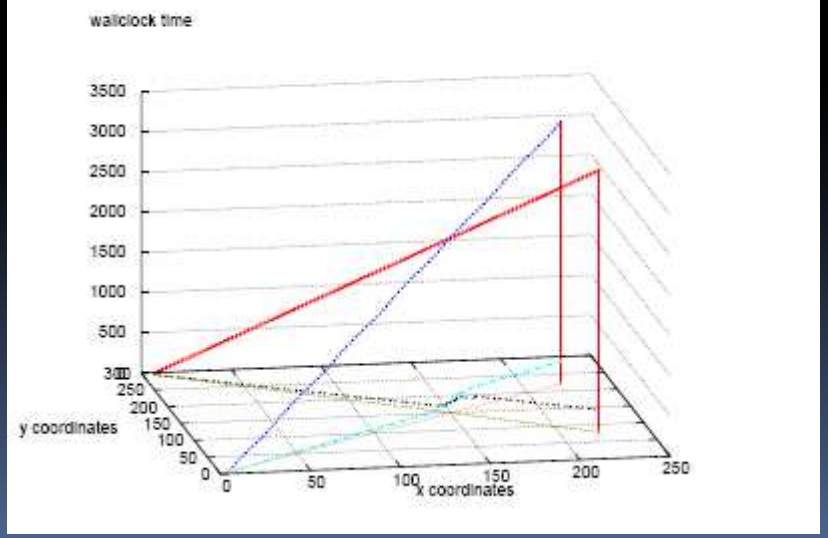


Fig. 9. Player position versus game frames. Top lines:players A, B, C, and D. Bottom lines: paths in the xy-plane.



AS Protocol

1. 다음 턴에 플레이어의 게임영역에 침범하게 되어 영향을 줄수 있게 되는 sphere의 영역.

2. 두명의 플레이어가 다음턴에 서로의 영역을 침범하지 않는다면 결정적인 이벤트에 대해 서로 영향을 미치지 않게된다.

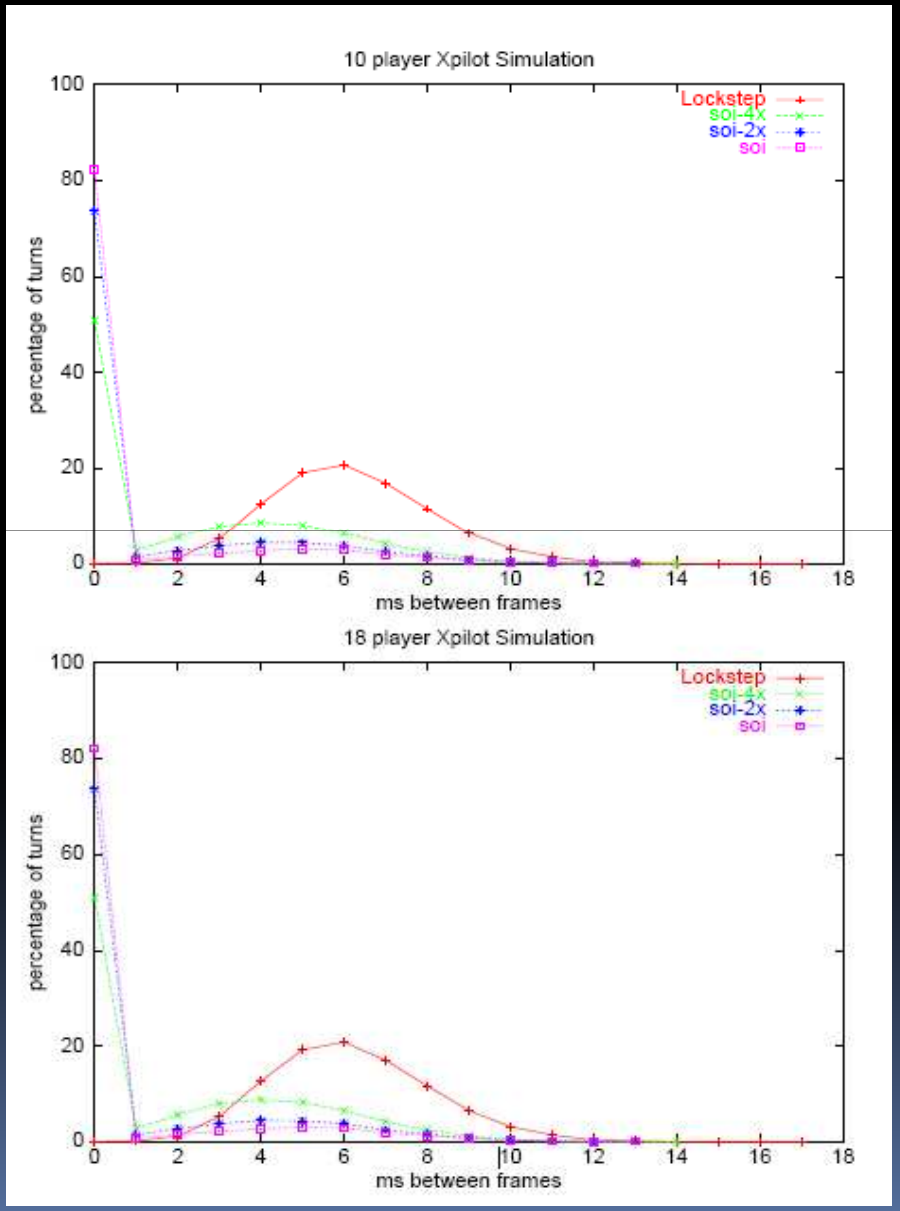
3. E.g.) 플레이어가 숲속의 소리가 닿는 영역에 있지 않다면 떨어지는 나무가 만들어내는 소리에 대해서 고려할 필요가 없다.

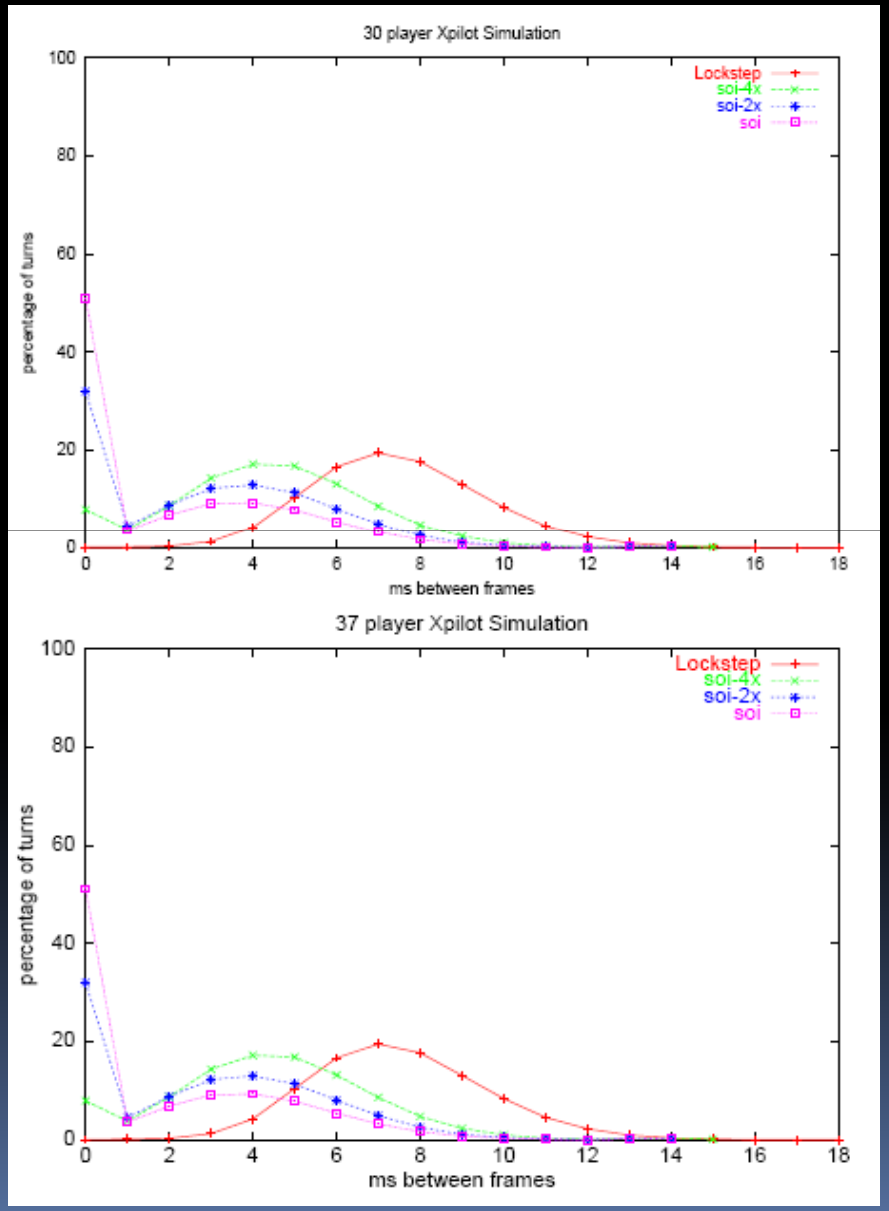
AS With Packet Loss

1. AS의 증명을 통한 모든 플레이어는 신뢰된 채널위에 존재한다. 플레이어들은 잃어버린 패킷을 스킵하고 다시 받을 수 있지만 버리거나 그냥 넘길 수는 없다.

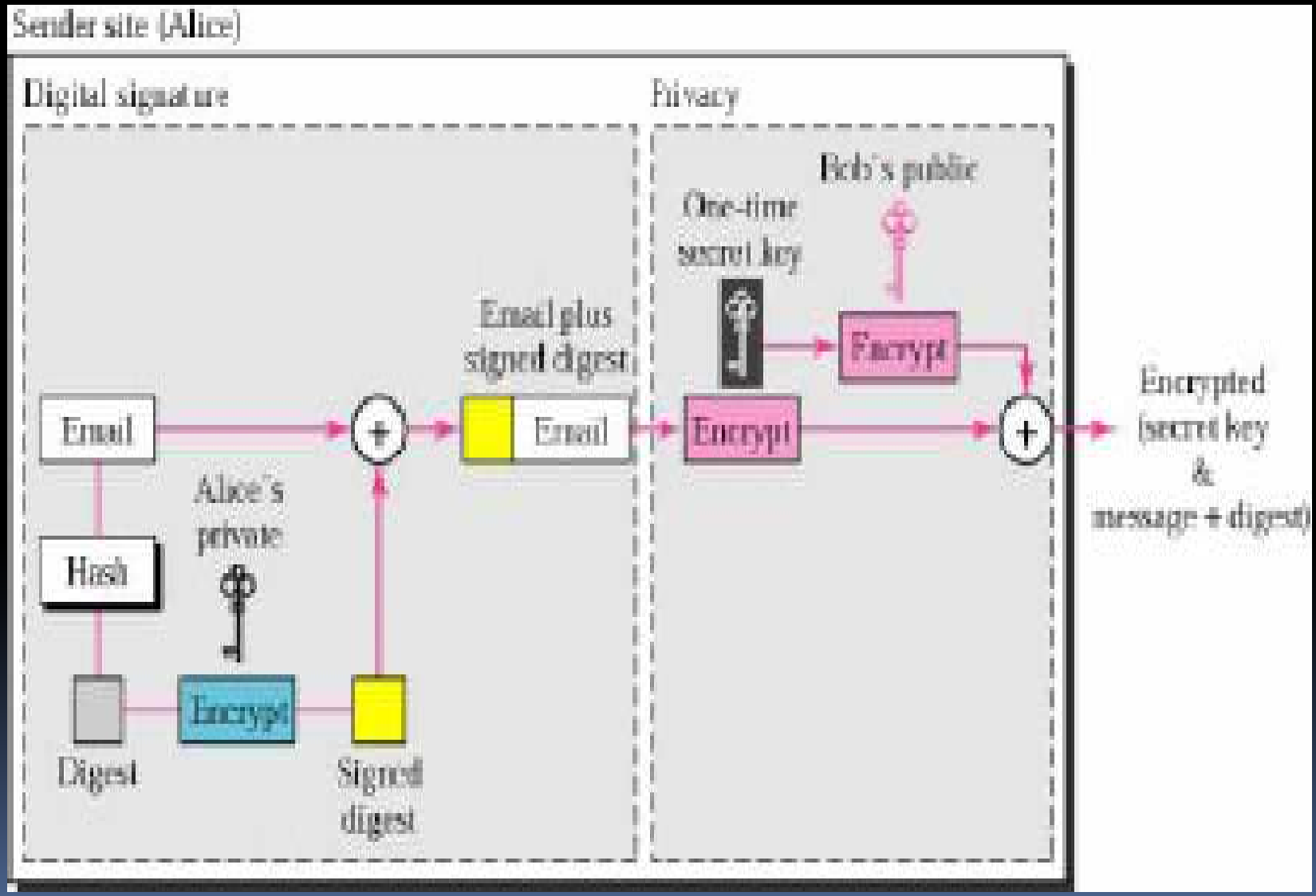
2. 이 때문에 SOI영역 밖에 있는 모든 플레이어에게 패킷을 받는 여부와 관계없이 모두 전송되게 되어지기 때문에 매번 각 플레이어는 모두 접근하게 된다.

Performance Analysis





Hidden Position



Hidden Position