9网络技术作业

Game	Genre	Network Protocol	Sync Method	Network Topology
StartCraft I/II	RTS	UDP	Lock Step	P2P
Warcraft I/II/III	RTS	UDP	Lock Step	P2P
Dota	RTS	UDP	Lock Step	P2P
World Of Warcraft	MMORPG	TCP	State Sync	Client-Server
League Of Legends	MOBA	UDP	State Sync	Client-Server
王者荣耀	MOBA	UDP	Lock Step	Client-Server
全民超神	MOBA	UDP	State Sync	Client-Server
Doom I/II	FPS	UDP	Lock Step	P2P
Quake I/II/II	FPS	UDP	State Sync	P2P
Counter Strike	FPS	UDP	State Sync	P2P
和平精英	FPS	UDP	State Sync	Client-Server
守望先锋	FPS	UDP	State Sync	Client-Server

问题：为什么他们会采取这样的方案，找找看有什么规律？

网络协议分析：

可以清楚的看到只有魔兽世界是使用TCP的，表里的其他游戏都使用UDP协议。原因在于这些游戏都是PVP游戏，着重于对抗与竞技，对于操作要求很高，因此游戏的流畅和实时关系到游戏的核心体验，因此选择可靠性差没有流量控制的UDP协议，这样虽然开发成本高（比如需要自己实现超时重传，keepalive等机制），但是避免了慢启动等TCP机制带来的弊端。而魔兽世界，作为以PVE为主的游戏，可以容忍一定的延迟，并且游戏中的系统多样，如任务系统，宠物系统等，所以同时需要保证可靠性，因此选择了适用于实时性和通信频率不高的TCP协议。

网络拓扑：

可以看到选择P2P拓扑的主要是以局域网为主的FPS和RTS游戏，这些游戏的定位其实属于单机游戏的一种，因此采用P2P拓扑可以避免部署没有中心服务器，从而为公司节约成本。
而采用CS拓扑结构的则主要属于线上收费的网络游戏，这些游戏一方面需要保证安全性，比如反作弊，防外挂，另外一方面游戏的运营也需要中心服务器，比如充值系统，活动系统等等。因此选择CS拓扑结构来使得控制权由服务器掌握，同时数据保存在云端，也降低了通信成本。

同步方法：

如图所示，可以看出FPS和mmorpg主要以状态同步（State Sync）为主，这是这些游戏可观测到的网络实体较少，具有比较好的性能，流量较高。并且容易做到预表现，断线重连也比较容易，而且离线重播的需求也比较少，而且该方法可以一定程度预防透视类外挂，这在射击类游戏中至关重要。

而RTS则以Lock Step方法为主，因为客户端可观测的网络实体比较多，如果使用State Sync则性能会比较差，因此选择采用Lock Step，这样流量只取决于网络玩家的数量了，其次开发起来也更高效，因为不需要前后端联调。
而moba类游戏则各有千秋，不管选用哪个方法都有自己的利弊，具体需要根据项目的人员组成，资源，定位去具体问题具体分析，比如王者荣耀采用Lock Step，一个好处就是生成回放视频的时候比较容易并且文件较小，可以让玩家能够在比赛完保存录像进行战局回溯，此处就不一一赘述了

	锁步同步	**状态同步**
流量	一般情况下较低，决定于网络玩家数目	一般情况下较高，决定于当前该客户端可观察到（Observable）的网络实体数目
预表现	难，客户端需本地进行状态序列化反序列化，进行Roll-Forth	较易，客户端进行预表现，服务器进行权威演算，客户端最终和服务器下发的状态进行调解（Reconciliation）和Roll-Forth
确定性	须要严格确定性	须要不严格确定性
对弱网络的适应能力	较低，因为较难做到预表现	较高，因为较易做到预表现
断线重连	较难，需比较耗时地进行快播追上实时进度的游戏状态	较易，服务器下发当前实时游戏状态的Snapshot即可
离线重播（比如播放录像文件）	较易，且重播文件大小较小（和流量相关）	较易，但重播文件较大（和流量相关）
实时重播（比如死亡重播）	难，视乎需求，客户端可能需要本地（性能消耗非常大地）每帧对进行全场状态序列化，从而能发序列化"回到过去"，并进行重播，播完后再（可能比较耗时地）快播追上实时游戏状态	较易，服务器下发历史Snapshot给客户端回到过去、下发重播数据进行重播、再下发当前Snapshot恢复实时游戏
网络逻辑性能优化	较难，因为客户端需要运算所有逻辑	较易，大部分逻辑默认是在服务器进行运算，从而分担客户端运算压力；服务器也可帮助客户端进行可观察网络对象的剔除（基于距离剔除、遮挡剔除、分块剔除等），也可以降低优先级低的物体或属性的同步频率，从而减小流量和再次减小客户端运算压力
大量网络实体时的流量情况	好，因为流量只决定于网络玩家数目	如果客户端可观察到的网络实体较少，则较好，比如PUBG等BattleRoyale类型；否则如果客户端可观测到的网络实体较多，则较差，比如Starcraft等RTS
大量网络实体时的性能情况	较差，因为客户端需要运算所有逻辑。如果大部分网络实体有"Sleep"的可能，则有优化空间	如果客户端可观察到的网络实体较少，则较好，比如PUBG等BattleRoyale类型；否则如果客户端可观测到的网络实体较多，则较差，比如Starcraft等RTS

外挂	因为客户端拥有所有信息，所以透视类外挂的影响会比较严重	也会有透视类外挂，但服务器会进行一定的视野剔除，所以影响稍小
开发特征	平时开发起来很高效，不需前后端联调，但写代码时需要确保确定性，心智负担较大，不同步bug如果出现，对版本质量是灾难性的	平时开发起来效率一般，需要前后端联调（LocalHost自测起来效率很高，但和最终Client-Server的真实情况不尽相同，自测应以后者为准，故依然需要联调），但写代码时不需确保确定性，心智负担较小，无不同步的bug
采用第三方库	较难，因为第三方库也须确保确定性	较容易，因为第三方库不须确保确定性