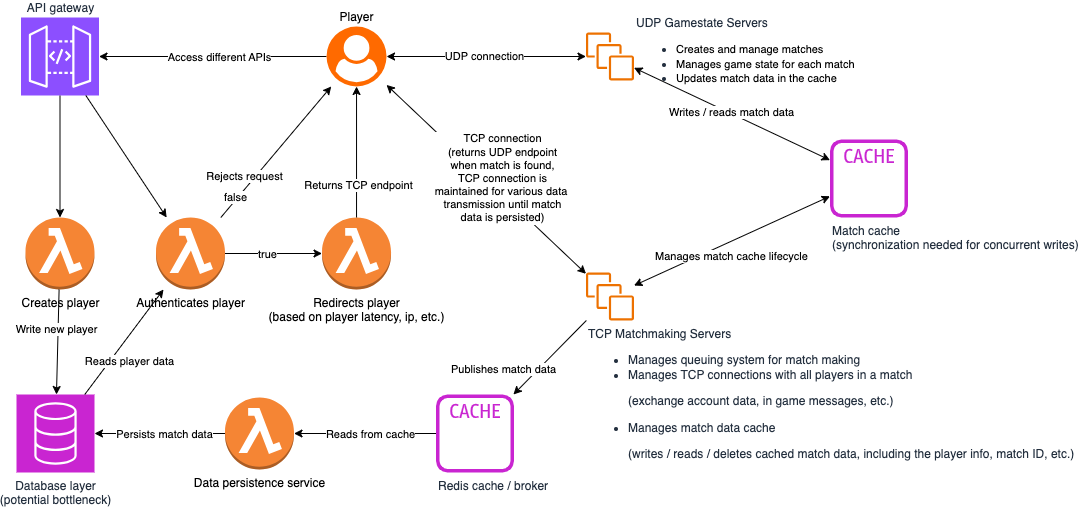
**CS6650 Bonus System Design**

Team RabbitRush

Xi Yang, Yuan Shen, Zhixuan Liu, Nahai Gu

**1. System Overview**

This MOBA game architecture uses a multi-layer, protocol-optimized design to balance real-time gameplay with reliable matchmaking and data persistence.



Key components include:

**API Gateway**

* Routes requests to appropriate services (matchmaking, gameplay, auth).
* Enforces security and load balancing.

**TCP Matchmaking Servers**

* Handle player queues, authentication, and TCP-based pre-match communication (e.g., lobby chat, player data sync).
* Write initial match data to a **sharded cache**.

**UDP Gamestate Servers**

* Host real-time matches with low-latency UDP for game state updates.
* Caches the most time-sensitive game data (graphic rendering data based on on-going events, etc.).
* Continuously sync match state to the cache.

**Cache Layer (e.g., Redis)**

* Caches less time-sensitive game state data (final game statistics, player rank updates, etc.)
* Acts as a single source of truth for player information in active matches.
* Requires synchronization for concurrent writes

**Data Persistence Service**

* Asynchronously saves finalized match data to the database.
* Can scale up / out based on load.
* Uses a pub/sub model (RabbitMQ, Kafka, etc.) to decouple writes.

**Database Layer**

* Stores persistent data (player profiles, match history).
* Potential bottleneck under high write loads.

**2. Collaboration**

The system architecture was developed through a collaborative, iterative process, integrating feedback and insights from all team members. Additionally, the design incorporates real MOBA game gaming experience and discussions with other students within and outside the classroom.

**3. Key Tradeoffs**

**Table 1. Design Tradeoffs**

| **Design Choice** | **Advantages** | **Challenges** |
| --- | --- | --- |
| **TCP for Matchmaking, UDP for Gameplay** | Reliable lobby setup and low-latency gameplay.  Independent scaling (e.g., more UDP servers for peak hours). | Complex TCP→UDP handoff; UDP packet loss risks.  Cross-service coordination overhead. |
| **TCP vs. UDP, reliability vs. speed** | Separation of concerns. High scalability. | Uses UDP only but implements custom reliable transmission protocol on Application Layer for critical info |
| **Async DB Persistence** | Reduces gameplay latency. | Risk of data loss if cache fails before persistence. Might need to log the data in broker. |

**4. Conclusion**

This design prioritizes **real-time performance** for gameplay while accepting complexity in state management. It’s ideal for competitive MOBAs where low latency is critical, but requires robust:

* Data durability
* Failure handling (e.g., match data recovery)
* Monitoring for hot partitions