VampireTCP Server Design Sheet

1. Architecture

1.1 High-Level Components

- **Configuration Manager:** Reads and validates a JSON configuration file to set up TCP/UDP ports and default room settings.
- **Networking Layer:**
  - **TCP Server:** Listens on a customizable port and handles client connections.
  - **UDP Server (Optional):** If enabled, binds to an additional port for high-frequency, non-critical data packets.
- **Room Manager:** Manages room creation, deletion, and tracking. It distinguishes between public and private rooms and provides APIs for room listing.
- **Room Class Hierarchy:** Provides a generic Room class with default implementations. Developers can extend this class to create Custom Room classes, overriding methods like `onJoin` and `onLeave`.
- **Event System:** Monitors room variable changes and triggers broadcasts to all clients in a room if the variable's update flag is enabled.

2. Detailed Design

2.1 Configuration

The server configuration is handled through a JSON file. A sample schema is provided below:

```
{
  "tcpPort": 8000,
  "enableUDP": true,
  "udpPort": 8001,
  "defaultRoomSettings": {
    "public": true,
    "variableBroadcast": {
      "enabled": true,
      "default": false
    }
  }
```

```
}
```

The configuration manager validates required fields and assigns default values for missing settings.

2.2 Networking Layer

- **TCP Server:**
  - Reads the TCP port from configuration.
  - Initializes a non-blocking server socket.
  - Accepts client connections and passes them to session handlers.
- **UDP Server:**
  - Checks the `enableUDP` flag from the configuration.
  - If enabled, binds to the designated UDP port.
  - Processes high-frequency, non-critical packets with lower priority compared to TCP.

2.3 Room Management and Class Hierarchy

- **Room Manager:**
  - Provides an API for room creation: `createRoom(customRoomClass?, roomSettings?)`.
  - Maintains an index of all rooms with public/private flags.
  - Offers a method `getPublicRooms()` to retrieve all public rooms.
- **Generic Room Class:**
  - Default methods include `onJoin(client)` and `onLeave(client)`.
  - Handles variable management through `setVariable(name, value)` and `getVariable(name)`.
  - Supports method invocation via `invokeMethod(methodName, args)`, returning results to clients if necessary.
- **Custom Room Class:**
  - Developers can subclass the Generic Room to implement custom logic and override default behaviors.
  - **Example:**

```
class CustomRoom(GenericRoom):
    def onJoin(self, client):
        # Custom join logic
        super().onJoin(client)
        # Additional processing here

    def customMethod(self, arg1, arg2):
```

```
        # Process custom request and return value
        return result
```

2.4 Variable Events and Client Communication

- When a room variable is updated via `setVariable(name, value)`, the method checks if it is flagged for broadcast.
- If the broadcast flag is enabled, the new value is sent to all connected clients in that room.
- Communication uses TCP for critical commands (like room creation and notifications) and UDP for high-frequency updates.

2.5 Overriding Join/Leave Behavior

- The Generic Room provides default `onJoin(client)` and `onLeave(client)` implementations.
- Custom Room classes can override these methods to implement tailored behavior such as sending welcome messages or updating room metadata.

3. Implementation Considerations

3.1 Concurrency and Thread Safety

- Utilize asynchronous I/O or a multi-threaded event loop to manage simultaneous connections.
- Ensure thread safety for shared resources such as room states, variables, and client lists (using locks or atomic operations as necessary).

3.2 Error Handling

- Perform thorough validation of the JSON configuration at startup.
- Gracefully handle network errors, including dropped UDP packets and unexpected TCP disconnections.
- Provide clear error messages for invalid operations, such as room creation errors or invalid variable updates.

3.3 Extensibility and Future Enhancements

- **Security:** Implement authentication and authorization for sensitive operations (e.g., joining private rooms).

- **Logging and Monitoring:** Integrate a logging framework to monitor key events such as room creation, variable updates, and error conditions.