Transparency Layer
Restructuring Transport Layers for Enhanced Operation

Abstract
A network transparency layer is the ability of a protocol to transmit data over the network in a manner which is transparent (invisible) to those using the applications that are using the protocol. Currently, the need for transparency layers have grown exponentially over the years. As businesses grow larger and larger, so do their programs. This can create a clutter of excessively large data collections on their client, which a large majority of the time, cannot handle such large data without slowing down the system immensely.

By designing a fully functional database using skills in Node.js and SQL, we proceeded to design our own transparency layer protocol that allows the user to easily manipulate data back and forth when the client requests it from a large scale program. Our transparency layer specifically focus around JavaScript's ECMAScript 6, which will allow businesses who utilize our transparency layer to easily create large scale JavaScript programs that run client sided but don't use all of the client's memory.

Results
Our transparency layer provides multiple uses and results for the end user. We create an environment that is more friendly and malleable to excessive recreation and data manipulation. Below are a few benefits represented in our transparency layer:

- **Caching**
  - Utilizes deep-depth caching algorithms.
  - Quicker response time visually and database-wise.
- **Core Library**
  - Includes functions to utilize various data sets, that are malleable and can extend to any use case.
  - Allows for easy porting to other systems.
- **Extendable**
- **Optional GUI-based Subsystems**
  - Functionality targeted towards GUI based designs.
  - Easy implementation to procure data in order to draw objects to a screen for GUI design.
- **Top-level scale ability**
  - Advanced algorithms and data structures to allow for large level scale if needed.
  - Hundreds of thousands of potential endpoints.
- **High Demand**
  - Focuses on specific market holes where no solution currently exists.
  - JavaScript's ECMAScript 6 pipelined via Autobahn's WebSocket protocol.
- **Large Accessibility**
  - Multiple users able to procure, edit, and manage the database simultaneously.
- **Autobahn**
  - Publish and Subscribe (PubSub).
  - routed Remote Procedure Calls (rRPC).
  - Multi-level support.
  - Ability to run in browser, or even in Node.js environments if needed.
  - Ideal for distributed, multi-client and server applications such as the internet of things applications or multi-user database-driven business applications.

Architecture

<table>
<thead>
<tr>
<th>Database</th>
<th>Web Socket</th>
<th>Caching</th>
<th>JavaScript Application on top of ECMAScript 6 API</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL/Node.js</td>
<td>Core Features</td>
<td>Large data pool</td>
<td>Auto-bahn, Large Accessibility</td>
</tr>
<tr>
<td>Multi-level</td>
<td>Stream utility Functions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Impact
The end goal of this project would be to have a replicated database within the client's host computer that stores the commonly used functionality of the program, and transmits certain function calls and data to the database when certain sets of data become unnecessary for the client to hold, or when the client's computer begins to suffer from too large of a data set. By caching the specific elements, we hope to accomplish a more discrete system of transparency unlike that of similar solutions.

Along with the experience gained from pursuing this project, the demand for transparency layers has skyrocketed over the years. By focusing on JavaScript applications, our usability for this project specifically targets that subset of current in-use application domains, allowing us to narrow in and focus on a specific point that can be improved and provide a better solution than any current existing ones.

The overall impact from all of this is a faster, friendlier, more maintainable, efficient, and straightforward transparency layer for the market to use.

Metrics
One of our design goals was to create a certifiable way to compare designs that utilize our transparency layer with designs that do not.

We utilized our test bed's interface to easily display a timer that properly calculates the time differential between runs, allowing us to easily see how much time was saved using our transparency layer.

Using this method, we saw about a one second decrease in query time when using our design, versus when our design is not in place.

In larger, more complex data structures, we would expect that we would have a more significant impact in time differentials, scaling up to larger, more industrial grade designs.

In the future we plan to increase our test bed exponentially, hopefully allowing us to see a non-linear decrease in query time.

Future plans and final notes
Moving forward, we plan to continue support for our product under the guidance of a local company.

Furthermore, via the guidance of Dr. Çankaya, the possibility of writing a research paper on this design has been discussed and is being considered.

The team would like to thank our main faculty adviser, Dr. Çankaya, along with our secondary faculty adviser, Dr. Razo. Throughout the project, we were helped with multiple ideas and online sources to guide us through this project, provided by professor Çankaya, where, without them, we would not have been able to produce a T-layer of this quality.

CS 4485 / Spring 2017
Department of Computer Science
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas
Richardson, Texas 75080, USA