Load Balancing for Web Socket Connections

Paul Oldridge
Department of Computer Science
Faculty of Graduate Studies
University of Victoria

April 15, 2015

Intro

Load balancing is a well established best practice for scaling to handle large amounts of traffic in the client/server model. Both TCP and HTTP load balancing are common in production environments. Typically load balancing is used for short-lived connections, but Websockets are long-lived. In this paper we will explore how to gain the advantages of load balancing, for long-lived websocket connections.

Background

Load Balancing

Load balancing is simply any means of balancing traffic amongst multiple servers. It is used to enable horizontal scaling, which improves both scalability and fault tolerance. The simplest, and cheapest, form of load balancing is round robin DNS, which occurs during DNS lookup and therefore requires no additional hardware. Round robin DNS, works by configuring the nameserver to respond to each DNS request it receives with the addres of the next server in the pool. This distributes load to all the servers, but not necessarily in a fair way; one machine could receive several requests which take large amounts of processing, while another receives small requests which require almost no processing. According to the law of large number, the imbalance tends toward zero as the number of requests gets large. Amazon Web Services (AWS) Route 53, extends round robin DNS with health checks, which allow the DNS server to remove addresses from its pool if the machine becomes unresponsive. Unfortunately, DNS queries are often cached around the globe for up to 72 hours, ignoring the TTL that was included in the query response.

A more comon approach to load balancing is to have a machine, or cluster of machines, dedicated to the purpose. These machines perform little to no processing on their inputs and simply forward packets to the actual servers. They also provide an added layer of security by being the only nodes in the network that need to be accessible from the internet. The actual web servers can all be within a private subnet, and only respond to messages from the load balancer. Most load balancers are transparent to the client, so the client doesn't know whether they are connected to a load balancer or an actual server. Bryhni, Klovning, and Kure [6] stated that application layer load balancers are often not scalable, since they can easily become the bottleneck. However, since 2000, there have been major changes in the role of a typical web server. We have transitioned from web sites to web applications, where the main purpose of the server is no longer to server static files.

Advantages of having a machine dedicated to load balancing, include SSL termination (the client connects to the load balancer over ssl, but the internal forwarding from the load balancer to the server is over plain TCP), better balancing algorithms (least-connections, least load, IP hash), and application layer balancing configurations, such as sticky sessions.

The most popular type of load balancer are software based, and usually run on Linux machines, for example, HAProxy, Nginx, Apache, Pound, Balance, node-proxy, etc... Enterprises managing their own data centers will often have dedicated load balancing hardware, which comes with proprietary load balancing software and algorithms. Companies which sell this type of hardware incluse RadWare, Cisco, and Barracuda.

AWS Elastic Load Balancer (ELB), is a cloud based fully managed load balancing solution by Amazon, which uses physical load balancers managed by Amazon to scale to handle any amount of traffic (Amazon has the infrastructure in place to add capacity well before it is needed).

Web Sockets

The HTML5 websocket specification, allows for browsers to form direct TCP full-duplex connection to servers. This, as well as another HTML5 specification, Server Sent Events. solves the problem of allowing browsers to receive notification from servers when their is new data for them to display. Previously we had to resort to polling, using the request/response model. Websockets also have minimal overhead, with a frame of about three bytes per message. When compared to HTTP, which typically has a header length of 200 - 800 bytes per message, [5] it is obvious why websockets are the preferred solution for real-time web applications.

Concerns have been raised about the design of the web socket protocol, in particular with regards to how they can achieve enterprise-grade scalability [4, 10]. Websockets are long-lived TCP connections, but load balancing usually is applied only to short-lived connections. Load balancing a long-lived connection is not as valuable, because it doesn't provide as much fault tolerance, and it is difficult to predict load on a live streaming connection. In the next section we will compare various setups for load balancing web sockets.

Comparison of solutions

Use an ordinary load balancer, typically used for short-lived connections.

Pros	Cons
Able to use ELB, haProxy, and others, without any additional configuration.	
IP hashing algorithms are very fast [7], so there is little overhead.	
	AWS ELB, along with haProxy's least-connection balancing algorithm, will work for plain websockets, but not for socket.io, which requires IP hash balancing [2].
	Load balancer must keep the TCP tunnel open for the duration of the connection, which for a long-lived connection could cause the load balancer to become overloaded.

Table 1: Pros and cons of default TCP load balancing

Direct Server Return: Load balancer is only used to pick the machine to form a connection to, then a direct connection is made from the client to the chosen server.

Pros	Cons
The load balancer is only used for short-lived connections, like it was designed to be used.	We lose the security provided by keeping out servers in a private subnet. They must be exposed to the internet for this solution to work.
	The load balancer is not transparent to the client, and the client needs additional logic to handle reconnection attempts when it loses a connection.

Table 2: Pros and cons of Direct Server Return TCP load balancing

New Implementation

A node.js based load balancer, balancing individual messages, by terminating the websocket connection at the load balancer, and maintaining a pool of backend connections to each server. This solution can be extended with the Redis adapter for socket.io, which allows to use socket.io-emitter to send messages to the clients who are connected directly to the load balancer. The number of file descriptors on the machine running this needs to be increased to handle large numbers of requests, since each open request uses a file descriptor, as described by the websocket-bench README [3].

Pros	Cons
Balancing on a per message basis instead of per connection, gives us the benefits that come with a short-lived connection load balancer.	The load balancer must maintain a large number of open connections. The load balancer is a single point of failure.

Table 3: Pros and cons of Direct Server Return TCP load balancing

Redis 3.0.0, which was just released ten days ago (April 1, 2015) [1], as of the time of this writing, includes Redis Cluster. A distributed implementation of redis, that allows one to transparently horizontaly scale Redis. Many production environments include scalable installations of Redis 2.x, using custom solutions [9], or used a SaaS company like RedisLabs to manage Redis on their behalf.

Future Work

The implementation of a node.js websocket load balancer is very basic so far. In the near future we plan like to add additional algorithms for balancing, rather than just round robin. We plan to also extend the protocol, so that the backend servers can be notified when a new connection is made, or when a connection closes.

Health checks need to be added, so that the load balancer can discover new servers, and remove unhealthy instances from its pool.

We would also like to explore a three layer model, where the first layer is ELB, the second layer is the new node.js load balancer we implemented, and the third layer is the actual web servers. This could allow us to overcome the only cons of this solution, by having many of these node.js load balancers we remove the single point of failure, and can scale to handle any number of connections, by adding more load balancers. Each load balancer within a region can connect to all of the servers for that region.

Bibliography

[1] Redis 3.0 release notes. `https://raw.githubusercontent.com/antirez/redis/3.0/00-RELEASENOTES`. Accessed: April 11, 2015.

[2] Socket.IO: Using multiple nodes. `http://socket.io/docs/using-multiple-nodes/`. Accessed: April 12, 2015.

[3] Websocket bench. `https://github.com/M6Web/websocket-bench`. Accessed: April 15, 2015.

[4] Amir Almasi and Yohanes Kuma. Evaluation of websocket communication in enterprise architecture, 2013.

[5] M Belshe and R Peon. Spdy: An experimental protocol for faster web, 2011.

[6] Haakon Bryhni, Espen Klovning, and Oivind Kure. A comparison of load balancing techniques for scalable web servers. *Network, IEEE*, 14(4):58–64, 2000.

[7] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for internet load balancing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 332–341. IEEE, 2000.

[8] Ian Fette and Alexey Melnikov. The websocket protocol. 2011.

[9] Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735, 2012.

[10] Vanessa Wang, Frank Salim, and Peter Moskovits. *The definitive guide to HTML5 WebSocket*, volume 1. Springer, 2013.