



Executive Summary

The Internet has evolved and grown exponentially over recent years. So too have the demands and expectations for powerful, efficient and functional web based applications. Real-time data plays a critical role within these applications; whether checking sports scores, planning a journey, or just updating friends, real-time continues to be the expectation for today's web users.

Storing, consuming and serving data has never been easier and cheaper than it is today. With vast cloud based data centres, defined interoperability standards and advances in Internet architectures, today's world wide web offers unparalleled reach to new markets and customers.

At the same time, Businesses face significant challenges in building and deploying cost-effective large scale real-time solutions that deliver data and achieve interoperability; due to problems such as cross-browser incompatibility; cross-device accessibility, data management across limited or intermittent connections (such as mobile devices) and delivering data across geographies in an intelligent manner.

In this paper we examine some of these challenges and seek to provide an insight into how Diffusion – the real-time Internet communication platform from Push Technology – has responded by delivering scalable and efficient solutions that enable businesses to realise new opportunities in delivering data, and consequently build new relationships with their customers.

The Challenges

Architecting real-time, secure, Internet scale systems can be complex and expensive and poses many technical challenges for architects. Typically, issues stem from factors outside of our control, for example:

- What happens to the user experience when bandwidth is limited?
- What happens when a client connection is dropped or lost?
- What happens when a lost connection re-connects; what data should they receive?
- What connection transport should be used and how does this change across platforms and devices?
- How can large numbers of concurrent connections be supported and how can data be promoted across geographies?

“Architecting real-time, secure, Internet scale systems can be complex and expensive”

These types of issues are inherent when dealing with volatile data across unpredictable networks such as the Internet, and the challenges can be summarised as follows:

- Internet connections can be unreliable and disconnect and reconnect without warning
- Internet connection speeds can vary between different clients and devices
- Browser features and performance vary based on vendor and version intricacies
- Different Internet connected devices vary hugely in performance, form factor and technology support
- Scaling to support large numbers of concurrent client connections can be expensive
- Supporting real-time data across geographies is complex
- Low Latency, high performance and resource efficiency are critical to a good responsive user experience

Request-response and the benefits of Push Models

The Hypertext Transfer Protocol (HTTP) [1] is an application protocol that is the foundation of data communication for the World Wide Web. From the inception of the World Wide Web, the method for transferring information using HTTP was that of the request-response model. Request-response is a simple client server interaction whereby a client (in this case an Internet browser application) requests a specific web page (or any other resource). The web server then services the request (perhaps performing computational functions) and generates the response that is subsequently sent back to the client.

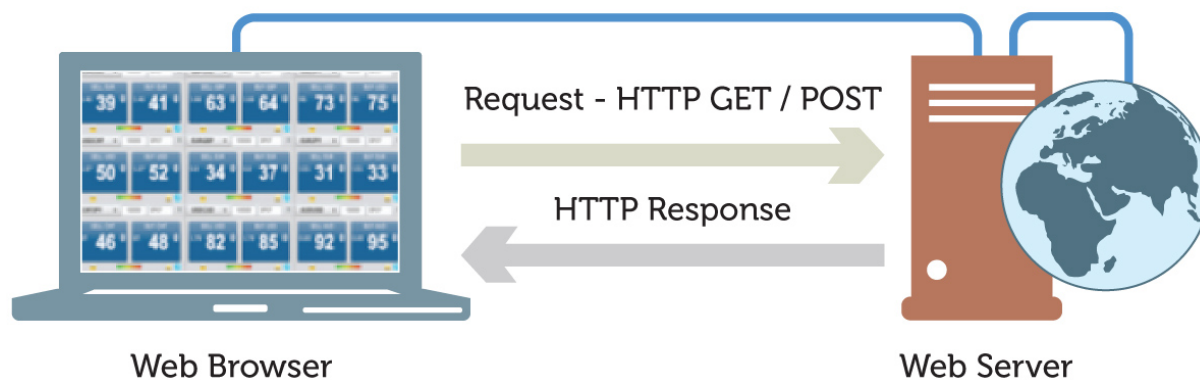
Whilst adequate for supporting the delivery of web resources such as HTML, CSS, JavaScript and other static media, fundamentally, the request-response model is inefficient for delivering real time data i.e. data that is volatile and always changing.

Consider a typical request-response based auction website where a web page may list 20 items whose prices are changing based on their respective bidding activity. In order for a website visitor to have accurate and up to date pricing information, the client will have to request the prices from the server at regular intervals - effectively asking the server “have the prices changed”.

There are two problems with this.

Firstly, a request and response may be made, even though the prices may not have changed. This uses system resources (CPU and computational resources along with bandwidth) even though no *new* data has been delivered. At scale, this can lead to significant unnecessary hardware costs.

Secondly, the requested data may become inaccurate and unsatisfactory, or stale, momentarily after the response has been received, therefore providing an inaccurate user experience. In the case of the auction website, the price may be out of date, so a new bid may be ineffective.



A typical HTTP request-response model for an Internet request.

As standards and browser technologies improved, techniques like AJAX (Asynchronous JavaScript and XML) were, and still are today, used to refresh only specific parts of the web page, offering periodic (e.g. every 30 seconds) and more frequent updates to individual data items, presenting the illusion of real-time data push. *These are however simply additional request-response calls, so in some use cases may reduce the problem of stale data whilst also placing further demands on hardware to support larger volumes of requests.*

Returning to the auction site example above once more, consider the page listing 20 items. Using a single request-response to deliver all 20 prices is done with a single call to the web server. Using an AJAX based page, each of the 20 prices may make individual calls to the server, for example, every 10 seconds. At 120 requests and responses per minute per client for a single page, it is easy to see how servicing these requests can easily become expensive with regards to hardware and bandwidth requirements.

Diffusion from Push Technology reverses the request-response paradigm, using a server *push* method to deliver real time data. This has significant advantages - only sending data to a client if and when data has changed thus offering an enhanced user experience where stale data is instantly refreshed. This method offers significant savings in both bandwidth and server hardware as the number of requests being serviced is dramatically reduced.

Advances in connectivity transports such as HTML5 with WebSockets, which are discussed later in this document, allow true, bi-directional communication for data push; offering high performance real-time connectivity using the Internet.

Real-time Data

At the core of any real-time data solution is the data itself. Modelling this data forms the basis of how the data is generated, managed and consumed so ensuring that the process is efficient and defined performance levels remain key.

Diffusion provides a high performance, low latency message broker for data management following the Publish/Subscribe [2] paradigm. This allows the data to be structured in a logical and semantic manner using *topics* and *topic hierarchies*.

Consider the following topic hierarchy:

/cars
/cars/audi
/cars/audi/petrol

Mapping topics to a hierarchical structure allows the consumers to very accurately segment and therefore select what data they require. Using the topic data example above, a consumer may choose to *subscribe* to the */cars* topic, meaning that they will receive data on this topic and all child topics i.e. */cars/audi* and */cars/audi/petrol*. Or, if they so choose, they could simply select */cars/audi/petrol* topic which would only deliver this specific topic data.

This publish/subscribe model provides a simple mechanism for fine-grained control over what is published and what is consumed by client and server.

A significant factor when selecting a message broker is the message format itself. Diffusion provides an **agnostic message format** that does not mandate a specific format or event type for the data. Diffusion provides optional field and record capabilities that can be utilised for convenience, however does not require data to be stored in a specific format. This is an important consideration when architecting integrations with other systems.

As well as being agnostic to a given message format, Diffusion is also **agnostic to the type of content**. This means rich binary data such as documents, images or even music can be delivered as easily as textual data. The message protocol itself is designed to be a low latency solution, mandating minimal overhead and therefore delivering high performance at scale.

Diffusion has **no built-in limit on the number of topics** that can be created, and given that each broker can subscribe to others, allows for exponential scalability and potential for growth. Architectures are discussed further within this document, however the broker subscription between nodes provides the ability to support broker or broker-less architectures, offering another degree of flexibility around data management.

HTML5 WebSockets

The concept of “data push” is not a new one. The push model is required for real-time data use cases, where making a *request* for some data, is actually requesting historic data.

The techniques used to implement push style capabilities typically rely on polling techniques such as *streaming* or *long-poll*. Comet is an umbrella term used for such polling techniques that allows long-held HTTP connections to be maintained, enabling real-time data to be pushed. These different implementations rely on technologies such as JavaScript and iFrames, and in-fact, are considered by some to be “hacks” of the traditional request-response model and the HTTP in order to meet real-time data requirements.

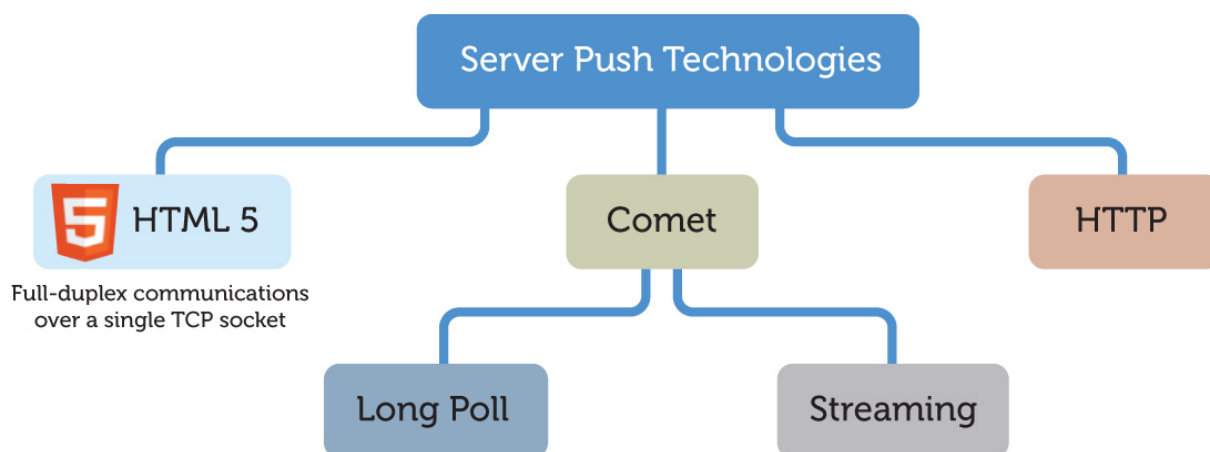
It is not the intention of this paper to detail the intricacies of each of these technologies however all of these implementations have, in some way, disadvantages. These problems and disadvantages include:

- Cross-browser incompatibilities
- Cross-site scripting restrictions
- Complex implementations
- Significant resource expenses caused by maintaining large numbers of open connections.

To provide full duplex communications allowing communication in both directions, and, unlike half-duplex, allowing this to happen simultaneously, two connections need to be maintained; server to client and client to server. In addition, such techniques all suffer with the overhead of HTTP, which does not make them suitable for low latency solutions.

HTML5 is the 5th iteration of the HTML specification [3] currently being developed by the W3C [4] and IETF [5] (at the time of writing, this specification is still under development).

The HTML5 standard includes many different enhancements in areas such as audio, video, and animation, and defined within the communications section of the HTML 5 specification is the WebSocket Protocol. The WebSocket Protocol offers a true full-duplex communication channel, capable of sending both a UTF-8 string and binary content in either direction at the same time, without opening multiple HTTP connections.



Summary of real-time data push techniques.

WebSockets offer a significant advance in web technologies delivering vast improvements in performance and latency. The WebSocket protocol uses the HTTP upgrade system (which is normally used for HTTPS/SSL) to "upgrade" an HTTP connection to a WebSocket connection, and as a result, a whole new dimension of communications is potentially available to HTTP based services.

WebSockets has established itself as a true bi-directional communications protocol and several web browsers already offer support for web sockets meaning the protocol can be utilised, even though the standard is not officially vetted.

This does however pose potential problems to architects looking to utilise WebSockets now. If the application in question is browser based, there may be client connections from browsers supporting WebSockets (such as Google's Chrome), as well as those accessing from browsers not supporting WebSockets (such as Microsoft's Internet Explorer). See [6] for a complete browser compatibility matrix.

*This is where Diffusion's **cascading transport mechanism** becomes essential.*

Upon requesting a connection from a client application to Diffusion using JavaScript, Diffusion will automatically attempt to utilise WebSockets as the transport mechanism. If WebSockets are not supported by a client browser, Flash Sockets are then selected as the transport. This graceful degradation of transports continues as compatibility is detected; to

Adobe Flash, then Silverlight, HTTP Ajax Long-Poll and finally to iFrames (supported by all major browsers).

From a developer's perspective, Diffusion's cascading transport mechanism delivers a significant saving in application logic. A simple connection call to Diffusion is made and Diffusion then takes care of selecting the appropriate transport. The order and availability of each transport is also configurable allowing for changes to order, as well as the removal of specific transports based on client application and audience scope.

It is clear WebSockets are a significant step forward in the evolution of Internet technologies. Diffusion's support for WebSockets, combined with its cascading transport mechanism allows developers to take full advantage of this technology today.

Cross Platform and Cross Device

Today's Internet is accessed by a huge variety of different platforms and devices, and this trend is only set to increase. New Internet capable devices are released daily - whether it's a new tablet device, mobile phone or an interactive or "smart" TV. Internet connectivity is also surfacing in less obvious applications, such as home appliances. These applications range from system and stock monitoring to live software updates, and integrations with social and ecommerce applications. The key message from these trends is that we cannot easily predict what devices and applications consumers of real-time data will utilise in the future, and as a result must build architectures with an agnostic approach.

Diffusion offers several key components to facilitate such architectures. Firstly, **Diffusion can support ANY net connected device**. As part of the core product, Diffusion offers 15 client side API's to make interaction and development as fast as possible. These API's include Java, .NET, Objective C (iPhone), Android, Blackberry, Flex, Flash, Silverlight etc. This list however does not restrict integration if a client side API is unavailable.

Push Technology has documented and published the Diffusion protocol, offering the opportunity for any 3rd party API to integrate with any connected device.

This is especially useful for hardware vendors where highly bespoke communications are required with the messaging service. Another key feature of building agnostic, or future proofing real-time, systems is that the Diffusion server sends the same data (or message) to any device or platform. **For Diffusion there is no difference in the data**. This means that the same data can be sent to an iPhone as is sent to a Java based desktop application. This offers significant benefits when considering expanding client side offerings.

Diffusion performs all of the message calculations and data intelligence on the server, leaving client applications effectively “dumb”. This has several advantages in that no client side calculations have to be performed e.g. calculating the current “state” of the data, or calculating deltas of change.

From a developer’s perspective, this represents a huge benefit. Developers can focus on implementing the GUI, and leave the complexities of the real-time data to Diffusion. This leads to a significantly reduced time to production.

Finally, the Diffusion message protocol itself is agnostic. Not only does it support any type of data, but it also does not mandate any specific format. Diffusion does offer formatting options (records and fields) along with helper methods for speedy data interaction, but if a custom or bespoke format is required, there are no limitations within Diffusion. This is a compelling attribute for any organisation considering integrations with current and legacy systems.

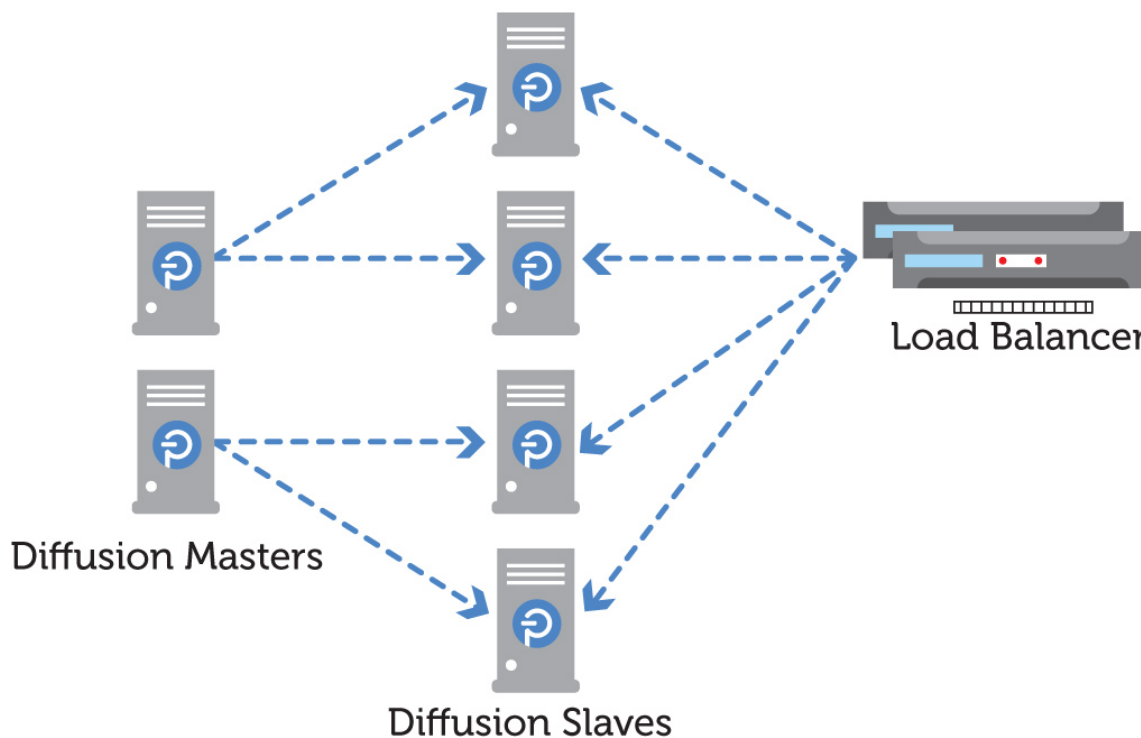
Scalable Client Connectivity

One of the other key challenges businesses face in streaming volatile real-time data is how to support large numbers of concurrent client connections in an efficient and cost effective way. The scalability constraints and limitations of many web servers is typically due to a finite resourcing limit, which many solve by deploying additional hardware. This rapidly accelerates costs, not just on the hardware itself, but also in continued maintenance and support, and the bandwidth costs of servicing large scale request-response architectures.

The aforementioned resource limitations of a web server can be explained as follows; a web server will establish a TCP/IP socket using a single thread, or worker resource, and will then wait to service an incoming request as each incoming request is passed to a pool of resources (worker thread pool). Operating system processes however can only maintain a relatively small number of threads, and if too many are created, the operating system will spend more time managing the threads than processing the requests. As a result, there is a limit to the number of simultaneous requests that a server can handle.

Diffusion breaks this paradigm by servicing and caching incoming connections, effectively allowing them to sit dormant until new events arrive. Diffusion uses client multiplexers to effectively “smart dispatch” messages based on virtual message references, so messages are never copied. This allows very large numbers of connections to be supported with high message throughput from commodity hardware.

Diffusion also offers several architectural options when scaling to support large volumes of concurrent connections. Diffusion servers can subscribe to each other, allowing exponential scale-out to support larger connection requirements.



Scaling Diffusion servers to support high volumes of client connectivity

Geographic Dispersal

The challenge of supporting real-time data at scale across geographies can be a complex one. Consumers of such data expect their data service to be robust and reliable (highly available) and most importantly consistent, irrespective of physical location.

In business critical scenarios where low latency is key such as financial trading environments, real-time data must be delivered and published in a synchronised manner. If this synchronisation is not available, or not accurate, critical pricing information could be available to certain audiences before others, leading to competitive advantage and potential for exploitation through arbitrage trading techniques.

Delivering real-time data across geographies can be broken down into three distinct requirements.

- Firstly, the solution must provide the ability to promote highly volatile data between data centres, maintaining performance; the key ingredient here being low latency.
- Secondly, the solution must be able to maintain “state” across geographies, providing a consistent view of the data, no matter where or how a client connection is made.
- Finally, as noted above, publishing the data in a synchronised fashion is key to delivering a consistent service.

Diffusion servers can subscribe to each other, allowing high performance data syndication between data centres and indeed geographies. Each server can subscribe either to the entire data set or a subset of the data, based on load and segmentation requirements. These dispersed architectures are yet further enhanced by support from the inbuilt “**State Engine**”. The State Engine allows any client connection to request a current snapshot of the data, with the data being *stateful* or *stateless* depending upon the requirements of the application.

Diffusion also automatically synchronises the publishing of the data across geographies, guaranteeing order, so that data can be published from different servers and locations at the same time.

Traffic Intelligence

There are many variables that can impact on the performance of real-time data solutions.

Performance can range from very fast, with vast data throughput at best, to intermittent, high latency, low bandwidth environments at worst, and everything else in-between!

Delivering effective, real time Internet solutions is not simply a case of connectivity. It is a combination of several components working seamlessly together, that when combined help mitigate risk and allow a good data quality of service to be maintained under fluctuating conditions.

“Device connectivity is only part of the challenge”

Diffusion was designed and developed to deal with unreliable operational conditions, and as such offers powerful capabilities for dealing with unpredictable shifts in service.

Diffusion supports variations in service and device providing the **complete end-to-end solution** for real time communications, consisting of the data source itself (high performance message broker), the connectivity gateway (supporting very large numbers of concurrent client connections) and the client side API's offering supporting features such as the cascading transport selection.

Unlike some products that only deliver a single component of this solution, Diffusion has the unique capability to oversee and therefore control the entire flow of real-time data.

At the heart of Diffusion's intelligent management solution are *virtual queues* for each individual client connection. Messages are placed on to each client queue (messages are never copied, only referenced for performance), and then placed on to the network interface card and sent for delivery. Diffusion monitors each individual queue for message activity. High and low watermarks can be set against queues allowing business decisions to be made based on activity.

For example:

If the message queue is building up a backlog of un-delivered messages and exceeding a high watermark, it is likely that this client connection or device is not able to cope with the supplied message throughput. Therefore, logic can be implemented to select a course of action. This may be to unsubscribe this client from specific data, perhaps low priority data not essential for the applications key purpose, or perhaps the decision may be made to disconnect the user all together, as the application experience may be unsatisfactory. The low watermark is used in the opposite conditions where it can be indicated that the client connection is experiencing good data throughput, and therefore could have specific data services re-enabled.

“Diffusion allows you to monitor, understand and make decisions for every client connection”

Using Diffusion's high and low watermark capabilities enables developers to make programmatic decisions based on device activity and performance. In addition, further traffic management features can be utilised such as **Throttling** and **Conflation**. Throttling limits the data throughput to specific clients where as Conflation provides the ability to effectively remove out-dated or stale messages within the client queue. Consider a high frequency trading environment where pricing data is being streamed. If a client queue contains a several prices for the same commodity, the Conflation facility will remove the stale data from the client queue, as there is little need for sending data which is out of date.

Using these intelligent data capabilities, Diffusion allows you to monitor, understand and make decisions for every client connection.

Coupled with the ability to monitor the performance of each client connection is the ability to manage **user sessions**. This extends the client management so that when connections are dropped, and re-connection occurs (perhaps from a mobile device), it is possible to decide what data that user receives. Perhaps the client requires that all missed data is sent for historical or auditing, or perhaps just sending the stateful snapshot of the data is preferable (i.e. *what is the state of the data now*). These options leave developers and architects in complete control of how their data is managed in these specific circumstances.

Summary

Real-time data services continue to play a significant role in today's Internet.

As the growth and availability of Internet connected devices continues to increase, so does the volume of real-time data generated each day. Low cost, high speed Internet connections coupled with accessible cloud-based infrastructures offer new levels of performance and scalability to Internet users, making real-time data an expected level of service.

Diffusion from Push Technology, delivers the complete, end-to-end real time data solution at scale, offering unique capabilities around data and architectural control.

References

1. [Hypertext Transfer Protocol](#)
2. [Publish Subscribe Pattern](#)
3. [HTML5 Draft Specification](#)
4. [World Wide Web Consortium \(W3C\)](#)
5. [The Internet Engineering Task Force \(IETF\)](#)
6. [Compatibility tables for support of HTML5 WebSockets in desktop and mobile browsers.](#)

