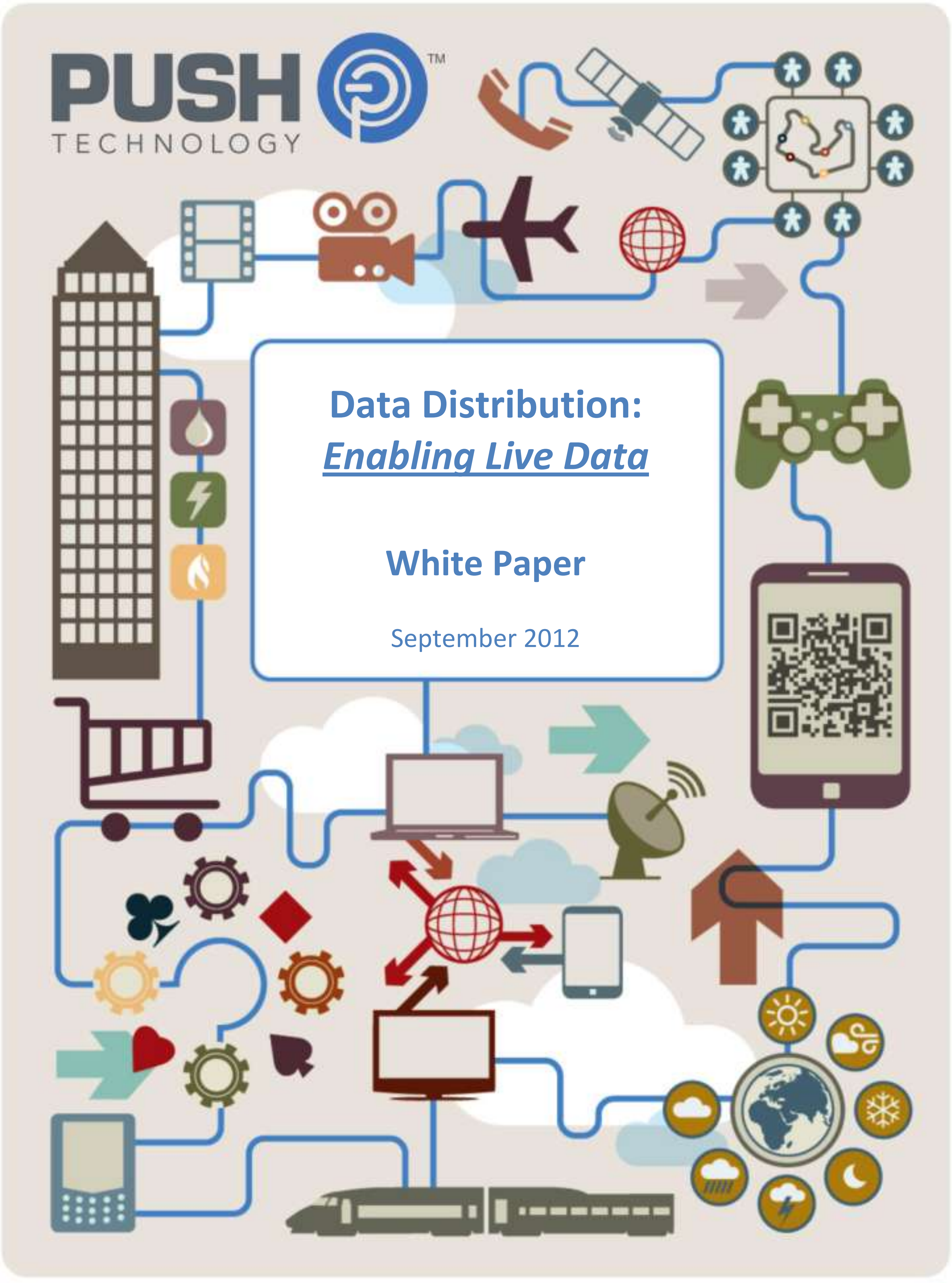


Data Distribution: *Enabling Live Data*

White Paper

September 2012

PUSH TECHNOLOGY LIMITED
107 CHEAPSIDE LONDON EC2V 6DN
Telephone: +44 (0)20 7397 2811
Facsimile: +44 (0)20 7397 2822
Email: info@pushtechnology.com
www.pushtechnology.com
Data On Demand

1 Introduction

Data Distribution – getting the right data, at the right place, at the right time – is a key challenge in today's increasingly connected world.

Rather than address data distribution by discussing messaging, this whitepaper will address it by discussing **live data**. Live Data is data that, by its nature, can change and it is these changes that consumers are interested in.

At Push Technology we believe that a data-centric approach is the key to solving today's data distribution needs.

2 Live Data

Live data is data that reflects the *most current information available*. By its nature live data is volatile and changes. The change may be fast or slow, but with live data only the most current information is valid. Consumers require that live data is up to date in order to make decisions. This contrasts with static data that reflects a single unchanging snapshot of data in time.

2.1 Live Data Examples

Some examples of live data include the following:

- Expected time of a train, bus, and or airplane
- Location of people, objects, or vehicles
- Live game state
- Live e-commerce auctions
- Trading or betting opportunities
- Traffic congestion on roads and other travel lanes
- Time or quantity-limited offers
- Inventory levels

Most of the application infrastructure today is focused on static data, and as a result most live data is treated as static data. Consumers are also used to this, e.g., needing to refresh the page to see if it's changed. However, as technology evolves and the network becomes omnipresent, expectations will rise. As the data becomes live, and consumers no longer need to hit refresh – data is always up to date – and those businesses that are unable to provide live data will be left behind.

2.2 Static Data Examples

Static data is data that is published once and *never changes in its current version*. There may be more up to date data as part of the context – a new version, new edition, and new history – but the specific data you are looking at will not change.

Some examples of static data include the following:

- A published edition of a book or article
- A legal contract or dated terms and conditions
- Technical documentation that relates to a particular version or model
- Historical data, analytics, or reporting
- Train timetable

2.3 Live Data: Today

The dominant model of the internet today is request-response. This is used for both static data and, frequently, live data. When this is used for live data, the response contains potentially out of date information. Critically, any changes in the data will not be reflected in the response!

There are a number of solutions in use today:

- Do nothing. This is by far the most common. Let the user refresh.
- Request-response polling. This uses a lot of bandwidth (polling when nothing has changed) and imposes unnecessary delays (waiting for next poll). This is only done for the most frequently-updated and sensitive live data.
- Messaging. Messages can be used to either update the client directly or notify the client to issue a new request-response. This is complex, potentially wasteful, and has a number of challenges. This is the most sophisticated solution and only used when absolutely required.

Doing nothing only works because *users are used to dealing with out of date data*. This is not tenable as an approach as users' expectations have changed.

The other approaches are complex, inelegant, and do not directly address the challenges of live data.

3 Live Data Challenges

Any live data solution involves active communication between the server (source of live data) and a client (consumer of live data). Building a live data solution has a number of challenges that need to be addressed.

3.1 Live Data for any client

Live data may be consumed on any number of clients, including mobiles, smart phones, tablets, desktop applications, browsers, thin clients, kiosks, smart televisions, and more.

Additionally, each client needs to be appropriately treated, i.e., authenticated and authorized for the relevant live data that can be consumed.

3.2 Live Data across any connection

Consumers of live data can be using any type of connection. The connection between the client and server can change in a number of ways:

- Bandwidth available
- Latency between client and server
- Connectivity itself; it may drop and re-connect, potentially with a new IP address

In order for a good experience by the consumer of the data all connection types – including no connection! – need to be accommodated as part of the solution.

3.3 Live Data at any scale

There are a number of ways in which the solution may have to scale that can have a dramatic impact on performance and user experience. These include the following:

- Number of connected clients
- Size of live data that is being shared
- Frequency of updates of live data

In order to maintain a good experience for the consumer the solution must scale to meet both the typical requirements as well as the peak requirements.

3.4 Live Data for any application

There are a number of ways applications, and the business they support, may require data to be modelled. Additionally larger technology infrastructure and business requirements can have a big impact. These include the following:

- Structure of data (lists, maps, embedded types)
- Semantic of updates (all updates important, only latest matters)
- Links between data (sub-data, complex structures)
- Persistence (JMS Queue,)
- Integration (In-memory cache, web servers, message bus)

4 Live Data: Current Technology

4.1 Request-Response Polling

Request-Response Polling has the benefit of using the existing infrastructure as-is, but suffers from several major issues:

- Polling uses a lot of bandwidth. Every poll is a new request-response. This is done even if nothing has changed.
- Polling is delayed to the frequency of the poll. Data won't be updated in the client any sooner than the poll will occur.

Due to the above, polling is not scalable and cannot be used freely. As a result, it is used infrequently and only when absolutely required.

Polling also consumes more power which is increasingly an issue for small mobile computing devices, particularly as data consumption for mobile usage grows.

4.2 Message Bus

Using a Message Bus is the most sophisticated approach available using currently established technologies. However, this approaches a data problem from a message-centric perspective and is similar to trying to solve a data storage problem using a file system!

4.2.1 Message Bus Complexities

The solution has a number of complexities that need to be addressed:

- Client state initialization. As a client starts, it must use the message bus to initialize its state. It could also use a request-response mechanism, but then the response must be synchronized with the messages.
- Client state updates. The client must subscribe to different topics on the message bus – which represent streams of messages – that then need to be mapped to client state.
- Message serialization. Message formats needs to be established and mapped to the state format and structure.

4.2.2 Message Bus Limitations

There are also some fundamental limitations in delivering data across any connection:

- Messages are opaque to the message bus. Messages cannot be dropped or conflated, as the message bus cannot have visibility into the meaning or context of the message. It may be a partial state update, for example. This means all clients must have the same minimum bandwidth, and higher bandwidth clients have no advantage. It also means that when the bandwidth drops below the minimum messages are queued and delayed!
- Re-connection. When the bus is re-connected, the application state must proceed from scratch. If there is any intelligence on re-connection (due to already received data), then it is up to the application to design.
- Messages buses assume a network is stable. This means that message buses therefore expect connectivity to be a constant.

5 Live Data: Future Technology

In the future Live Data will be managed in a data-centric approach rather than message-centric. This approach will be similar to current in-memory data caches, except it will be subscribable – clients will be notified of updates to data – as well as client facing.

5.1 Live Data: Subscribable

Current in-memory caches have a wide variety of features and options, but they all operate on a request-response approach. Live data requires asynchronous updates of data in the client, which is currently not how any in-memory cache works today.

5.2 Live Data: Client Facing

Current in-memory caches focuses on system-side concerns – server to server, and leaves the final client-facing concerns for other technology. Exposing a subscribable in-memory cache to end clients brings a number of issues to be considered:

- Client Scaling. How many thousands or millions of clients?
- Data Scaling. How much data and how dynamic is the data?
- Connectivity Options. What devices and transports do the clients use?
- Authentication/authorization. Who can see what information?
- Transport Security. How can we secure the data and server?
- Individual Client Views. How can each client get a custom individual view of data?
- Client Model Synchronization. How can we enable rapid GUI development?

6 Live Data: Diffusion™

Diffusion™ from Push Technology started as a messaging technology and has evolved into something much more. It now includes a data caching layer on top that allows for live data on the server to be intelligently distributed – using the core messaging technology – to any clients, across any connection, at any scale.

Push Technology is focused on building the next generation platform for live data on demand.

6.1 Diffusion™ Messaging Layer

Diffusion™ messaging layer supports a number of key features, including the following:

- Pub/Sub Hierarchical topics. This allows for clients to subscribe to individual topics, topic wildcards, or any combination in-between.
- Conflation. This allows for multiple messages to be logically combined into a single message for an individual client queue. Clients will always get the most up-to-date data possible given their bandwidth.
- Prioritization and message fragmentation. Large messages can be broken up and de-prioritized to allow for further control over client experience across bandwidth.
- Snapshot and deltas. Custom messages can be sent when a client first subscribes to a topic, while deltas are sent to all subscribed clients.
- Bidirectional. Messages can be received from the client that can be interpreted by the application logic.
- Customizable. Full APIs are available for customizing all aspects of the messaging protocol, including custom messages, gathering statistics, custom conflation, and more.

6.2 Diffusion™ Memory Cache Layer

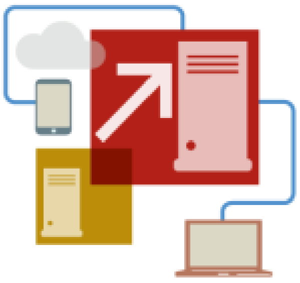
Diffusion™ memory cache layer allows data to be associated with topics. This supports some of the following features:

- Complex data structures. A number of different data types (Records and fields, Single Value, Protocol Buffers) are possible.
- Automatic snapshot and delta messages. As data is updated appropriate delta messages (if required) are generated, and new clients will receive snapshot messages.

- Automatic conflation. Data will automatically be conflated by the most appropriate way (combine deltas together).
- External Debugging. Diffusion™ includes the Introspector which allows a developer to inspect the current data cached in Diffusion™.
- Customizable. Custom data types are possible through available APIs.
- Easy to Use. Diffusion exposes developers to a publish/subscribe streaming paradigm, while supporting rich server side capabilities to ensure data is handled the right way.

6.3 Diffusion™ Client Capabilities

Diffusion™ has all of the Client APIs across a number of devices, including JavaScript, iOS, Android, J2ME, J2ME Blackberry, Java, C#, Silverlight, Flash/Flex, and C. These Client APIs include some of the following features:



- Connect using most performant transport. For the JavaScript API it will use the best transport of WebSocket, Flash, Silverlight, Comet, and Long-Poll. All other APIs also use the best available transport.
- Connect Securely. All APIs connect using username and passwords that can be authenticated by the server, as well as optionally use SSL as part of the connection.
- Custom Client Views. The server can optionally customize the view of each client, depending on the device, GeoIP, authenticated user, or any other criteria.

6.4 Diffusion™ Scaling

Diffusion™ has proven to be able to scale to the largest of websites. This includes

- Concurrent Connections: Individual Diffusion™ servers have been scaled to 150,000 concurrent connections, and Diffusion™ has been used to serve more than a million concurrent connections through horizontal scaling.
- Throughput: Diffusion™ has managed to saturate four 10Gb NICs on a single server and has managed more than nine million messages a second.
- Topics: Diffusion™ has scaled to millions of Topics, enabling fine-grained control of the client to determine the data of interest.

6.5 Conclusion

Diffusion™ provides a unique combination of a message bus and a data cache to enable data distribution of live data. By approaching the challenges of data distribution with a data-centric approach, rather than a message-centric approach, Push Technology through Diffusion™ enables a far more complete solution.

PUSH TECHNOLOGY

About Push Technology

Founded in 2006 and headquartered in London, Push Technology is an innovative and global technology and solutions specialist that focuses on projecting data beyond the edge. Uniquely capable of delivering Data on Demand, Push Technology's robust and scalable communication platform – Diffusion™ - enables large-scale smart data distribution that unlocks the true potential of today's multi-channel, connected world to help organizations realise meaningful, valuable and personalized exchanges.

With offices in New York, Maidenhead and London, Push Technology has hundreds of man years' experience working on mission-critical data and messaging solutions for leading companies in the Financial Services, e-Gaming, Social Gaming and Digital Media Sectors. It is this expertize and leadership in delivering Data on Demand, coupled with a proven track-record in successful implementations that has earned Push Technology its hard-won reputation for cutting-edge, reliable, scalable and high performing solutions.

Working with Push Technology, organizations with demanding real-time two-way communications requirements – such as online gaming companies, financial institutions, brokers, traders, spread betting firms and news feed providers – can successfully redefine the boundaries of their Internet performance.

For further information
Visit **www.pushtechnology.com** or contact **salesinfo@pushtechnology.com**

PUSH TECHNOLOGY LIMITED

107 CHEAPSIDE LONDON EC2V 6DN	**Telephone:** +44 (0)20 7397 2811	**Twitter: @push_technology**
340 MADISON AVE, NY, NY 10173	**Telephone:** +1 201-978-5574	**www.pushtechnology.com**