Reliable Client Session Management with Diffusion

1. Introduction

The adoption of Event-Driven Architectures (EDA) to build real-time applications is growing at an exponential rate. Pub-Sub based event-brokers are the popular choice among software architects and developers because the Publish Subscribe model enables event-driven architectures and asynchronous parallel processing, while improving performance, reliability and scalability. The real-time applications under development operate in environments with different network characteristics, and across platforms and/or geographies. Quite often the environments in which publisher and subscriber applications operate are not fully considered with respect to resilience. Software architects must evaluate the resilience and fault-tolerance features of any platform serving their applications.

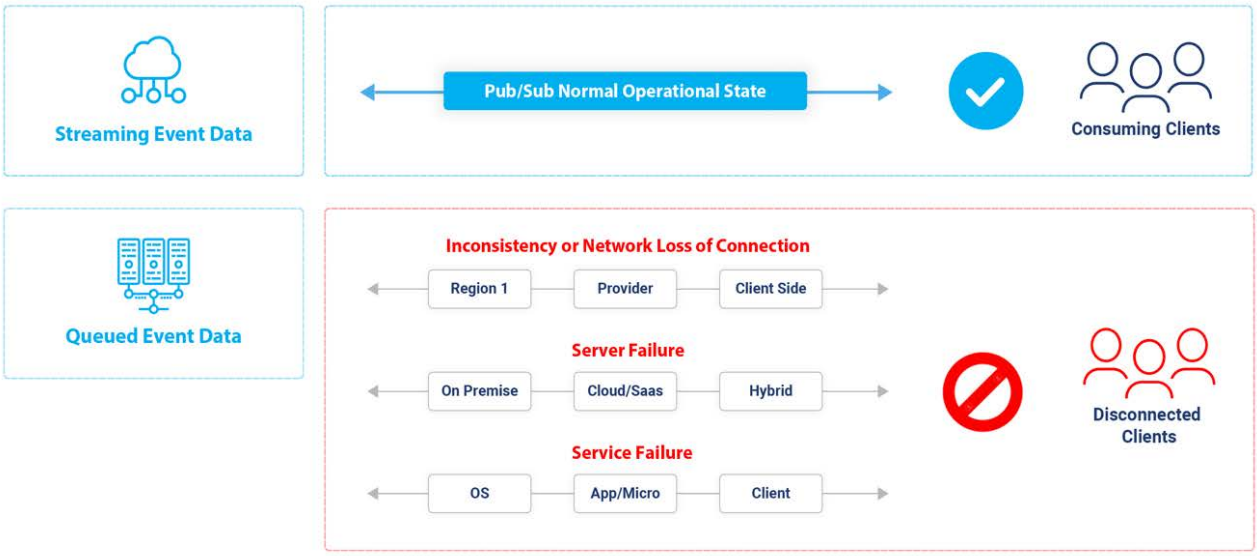
This whitepaper discusses the need for improved client session management in a pub-sub platform, followed by a description of the resilience features in the *Diffusion Intelligent Event-Data Platform that* deliver reliable client session management.

2. The Case for Client Session Management in Pub-Sub

In an event-driven architecture, using a Pub-Sub Event Broker, outages and interruptions can occur at several stages in the data consumption and distribution flow journey. For example, interruptions to applications on mobile devices may occur where the connection to a cell-tower is lost. Software architects are continually seeking ways to provide added resiliency for client applications to assure continued operation during and after such interruptions.

Below are examples of data streaming under normal operation and examples of situations where event-data delivery fails.

Figure 1 – Sample Event-Data Distribution Interruptions



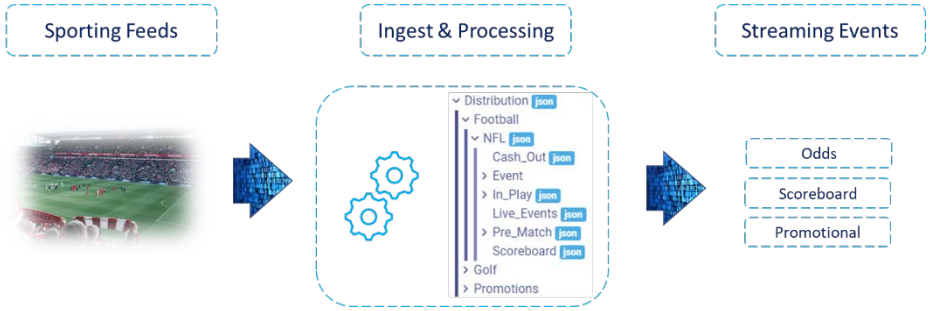
3. Diffusion Intelligent Event-Data Platform

Diffusion is a holistic pub/sub platform purpose-built to consume, enrich and transform, and deliver streaming event-data. Data can be ingested into Diffusion from any number of sources and data-stores, then enriched and organized as topics to which client-applications can subscribe to receive event-data. Event-Data is published into or subscribed from Diffusion using prebuilt or custom adapters and a rich library of SDKs/API(s) available in the popular programming languages including: JavaScript, Java, .Net (C#), Apple, Android, Python. Watch this video for a quick 60 second overview of Diffusion.

eGaming Use Case Example

A sports betting application can ingest sports data (directly from a sports league or multiple leagues) into Diffusion which is then processed and published as sportsbook odds, scoreboard data, and/or promotions in real-time.

Figure 2 – Sample Ingest and Event Data Preparation



Client applications can subscribe to Topics which contain the streaming event-data, such as the Topic Tree represented in *Figure 2*, subscribe to/Distribution/FootBall/NFL/Pre Match. The delivered event-data (message) could be:

Figure 3 – Sample Event Data

```
{
  "Home_Spread": "+11.5",
  "Home_Odds": "+120",
  "Away_Spread": "-13.5",
  "Away_Odds": "-140",
  "Home_Money_Line": "+375",
  "Away_Money_Line": "-530",
  "Home_OverUnder": "O 53",
  "Home_OverUnder_Odds": "+120",
  "Away_OverUnder": "U 53",
  "Away_OverUnder_Odds": "-140"
}
```

As event data consumption increases, additional burden is placed upon the network, servers, and services, impacting their ability to meet business defined SLAs and compounding the possibility of an outage. Using Diffusion, the burden is lifted from the network and back-end servers through the platform's patented delta streaming, conflation, throttling, delay, and other data wrangling capabilities.

4. Reliable Client Session Management in Diffusion

Reliable client session management is implemented in Diffusion in both client and server-side processes. As such, the following client resilience options have some correlating configuration and dependencies upon external factors including server settings and devices, such as load balancers.

When beginning to work with Diffusion to subscribe to events, client applications must:

1. Connect to Diffusion
2. Establish or connect to a Session
3. Subscribe to Event-Data Streams

Diffusion enables resiliency for subscribing clients in three distinct areas:

a. **Client Connection Parameters** define the server, security and resiliency properties for a client session with Diffusion,
b. **Session State Handlers** define custom actions when session states change (disconnect, reconnect, close, error),
c. **Data Delivery Resumption Features and Policy** for continuation of streaming events from the time of a disconnection and subsequent re-establishment of a lost connection.

a. Client Connection Parameters

Client applications connect to Diffusion via a rich library of Diffusion SDK/API(s) available in the popular programming languages including: JavaScript, Java, .Net (C#), Apple, Android, and Python. In addition, the SDK(s) provide embedded services at runtime for managing the connected sessions and queueing which are described later in this document.

Initially, connections are established with parameters, including those to improve resiliency.

Figure 4 – Diffusion Client Connection

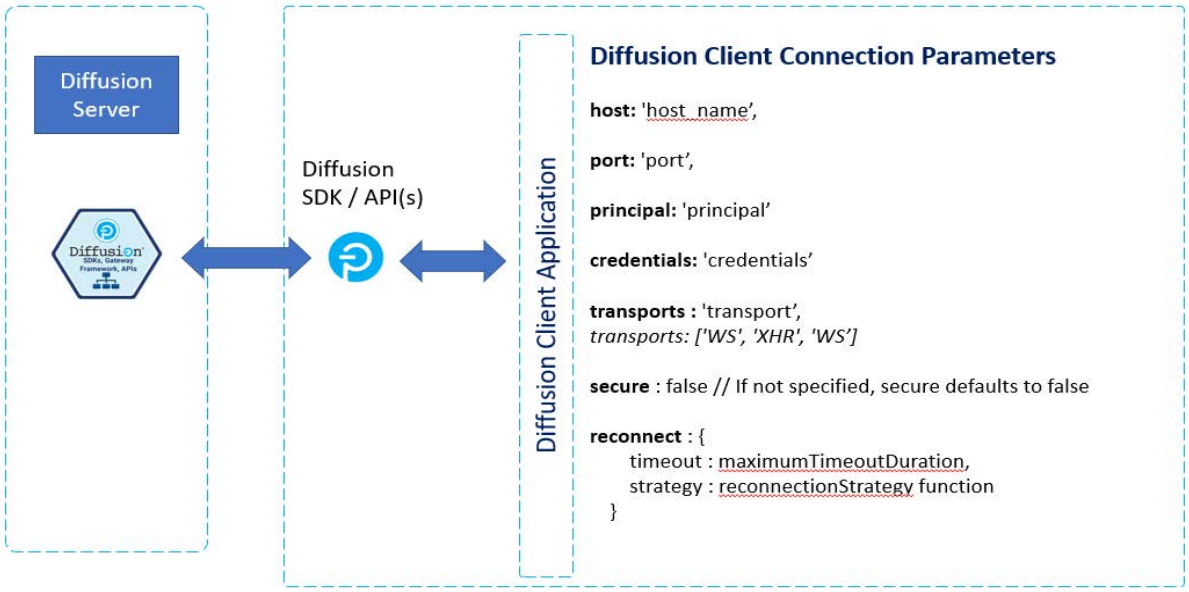


Figure 5 - A JavaScript example for a client to connect to Diffusion

```
diffusion.connect({
        host: 'host_name',
        port: 'port',
        principal: 'principal',
        credentials: 'credentials'
        transports : 'WS',
        reconnect : {
                timeout : maximumTimeoutDuration,
                strategy : reconnectionStrategy
                    }
}).then(function(session) { ... } );

…

function reconnectionStrategy(){
        // Custom Recovery/Alerting Code and Actions are placed
           here
}
```

Timeout – The Diffusion SDK will automatically continue to retry to connect to Diffusion when a client disconnection has occurred and until a timeout threshold has been met. A similar threshold is defined on the server. When a timeout threshold has been met, the server will delete the client session.

Custom Reconnection Strategies – The Diffusion SDKs also provide a framework and process for situations where a client application may be designed for extended processing when the connection is lost. A method can be defined for when a disconnected condition occurs, in the example above it is named as 'reconnectionStrategy'. This approach provides the greatest flexibility for custom processing in case of disconnections.

Pings – The Diffusion server will ping the client connections to ensure they are still connected over an interval (every 90 seconds is the default). In a similar fashion, a Ping service is available to the client applications for situations when they are aware of a disconnection/reconnection and want to ensure the Diffusion connection is still active. Below is a JavaScript example.

Figure 6 – A JavaScript example of how to ping the Diffusion Server Services via a Client Application

```
session.pingServer().then(function(pingResult) {
 // Take action based on ping details.
});
```

By default, a client will continue to try to reconnect to Diffusion in 5 second intervals over the period of the timeout until the timeout threshold has been met. To control the number of reconnection attempts or time intervals between, reference the example below:

Figure 7 – A JavaScript example for defining the number and timing of reconnection intervals

```
const MAX_RECONNECTION_ATTEMPTS = 5;
let attempts = 0;

diffusion.connect({
   host : host_var4.value,
   port : port_var4.value,
   principal : user_var4.value,
   credentials : "password",
         reconnect : {
                  timeout : 60000,
                  strategy : function(reconnect, abort) {
                           if (++attempts > MAX_RECONNECTION_ATTEMPTS) {
                                    console.log('Interval Test, aborting after: ' + attempts);
                                    abort();
                                    } else {
                                    console.log('Interval Test, connection try: ' + attempts);
                                             setTimeout(reconnect, 5000);
                                             }
                           }
         }
}).then(function(session)
```

b. Session State Handlers

Sessions are used by clients to interact with the Diffusion server, such as connecting to topics as well as publishing and subscribing event data. Sessions have properties specific to the client as well as objects for control and notification of event services. When a client is disconnected, the session continues to be queued on the server until the client reconnects or a timeout is reached. This logical abstraction of physical clients to diffusion sessions enables processing to continue in situations of physical interruptions.

In *figure 5*, parameters were listed as part of the Diffusion Connection followed by the creation of a session. The following statement then acts upon the established Diffusion Session.

Figure 8 – A JavaScript example of session processes

```
diffusion.connect({
         …
}).then(function(session) { … } );
```

In addition to Reconnection Strategies, the Diffusion client SDKs provide the ability to define callbacks for certain conditions of the connection, including disconnect, reconnect, close and error. These callbacks provide additional control for client applications to recover and act upon changes to external dependencies, such as a route to the server, or possibly server loss.

Figure 9 – A JavaScript example of defining action based upon Session state in a Client Application

```
session.on('disconnect', function(reason) {
      // This will be called when we lose connection. Because we've specified the
      // reconnection strategy, it will be called automatically when this event
      // is dispatched
    });

   session.on('reconnect', function() {
      // If the session is able to reconnect within the reconnect timeout, this
      // event will be dispatched to notify that normal operations may resume
      attempts = 0;
   });

   session.on('close', function() {
      // If the session is closed normally, or the session is unable to reconnect,
      // this event will be dispatched to notify that the session is no longer
      // operational.
   });

   session.on('error', function() { console.log('A session error occurred.');
   });
});
```

The code in Figure 9 is not dependent upon defining a reconnection strategy and can act independently, giving more flexibility to different changes in a client session. However, a reconnection strategy may provide further reconnection features. Selecting the optimal approach is dependent upon the use case requirements.

Note Client sessions may also be replicated to additional Diffusion nodes to provide added fault tolerance and prevent the loss of sessions due to failed hardware/services.*

c. Data Delivery Resumption Features & Policy

When a disconnected client reconnects to Diffusion, there may be need for the client to receive all the messages which have been published by other services since the disconnection. An example may be that the client is subscribing to stock ticker event-data from Diffusion and then loses its connection. Upon reconnection, the client needs the additional ticker data which has been updated to Diffusion from publishing services.

To resume the delivery of event messages upon reconnection, the Diffusion platform provides these capabilities.

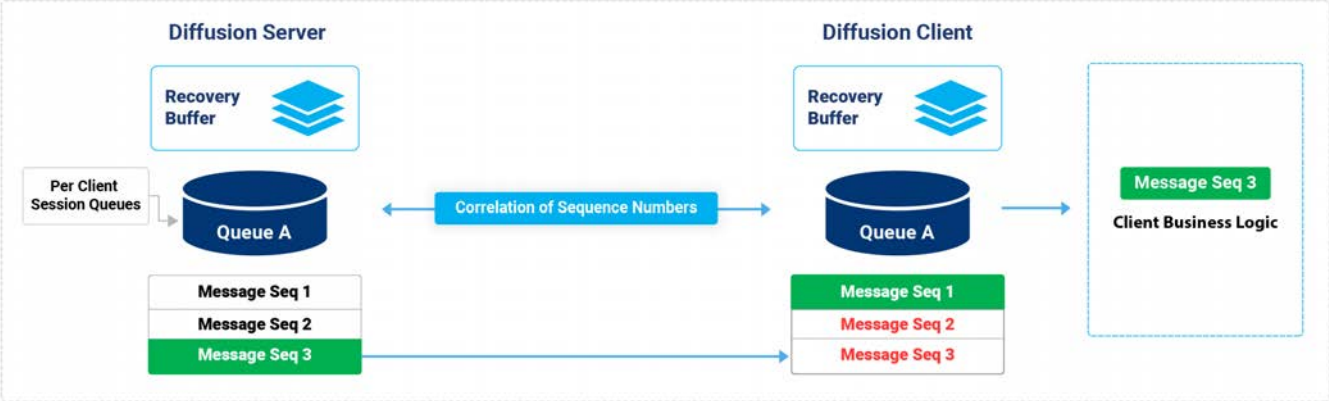
1. Server-Side Recovery Buffers
2. Client-Side Recovery Buffers
3. Event Message Synchronization Policies

Any messages (data) from Diffusion to clients are initially stored in server side queues for each client. If a client is unexpectedly disconnected, the queue continues to buffer event messages.

The queues are persisted unit the message(s) are delivered, or a configurable limit has been reached (the conflation property).

Siimilarly, the client tracks event messages which have been received.

Figure 9 – The default Diffusion behavior to resynchronize data for reconnected sessions



In the above example, the client connection is lost and then re-established. Messages continue to be published into Diffusion and are stored in separate Queues for each subscribing consumer during the disconnect (queue A for consumer A in this example). Upon reconnection, a synchronization occurs with the last messages between the Server and Client recovery buffers. In the above example, the message count on the server reached Sequence 3.

The client declares that the last message it received had the sequence number 1, so the server sends messages from sequence number 2 onward. If the server has already flagged a message as having been sent, it is retrieved from the recovery buffer and resent, otherwise it is taken from the client queue.

5. Conclusion

Client Application Resilience with Diffusion is just one category in the enablement of High Availability and Fault Tolerance for Event Driven Architectures. Please give us a call to learn about new features and best practices for implementing resilient applications with Push Technology's Diffusion.

6. Resources

Additional resources are available to explore and learn more about solutions powered by Diffusion.

Learn more:

- Learn more about Diffusion at the Product page
  - https://www.pushtechnology.com/product-overview
- View Interactive Demonstrations on the Diffusion Playground
  - https://www.pushtechnology.com/diffusion-playground/
- Review Diffusion Blogs
  - https://www.pushtechnology.com/resources/blog/
- Signup for a Free Trail of Diffusion Cloud
  - https://dashboard.diffusion.cloud/signup
- Contact us to speak to an expert
  - https://www.pushtechnology.com/contact-us-form

Video links

- Diffusion 3-minute Overview
  - https://youtu.be/Q-3Aor5Dt1o
- Data Wrangling with Topic Views in Diffusion
  - https://youtu.be/BC58KOpmnQE
- Low-Code features to build event-driven applications
  - https://www.youtube.com/watch?v=QdYG55Z9TNs&list=PLYzjzg_h2TczleO1yH9vbsoqmritzupGZ&index=2&t=12s
- Fundamentals of Authentication and Security in Diffusion
  - https://www.youtube.com/watch?v=cKugOluHUZI&list=PLYzjzg_h2TczleO1yH9vbsoqmritzupGZ&index=5&t=214s