Stringflow SDK Architecture & Design

(Build Version: 1.0.0)

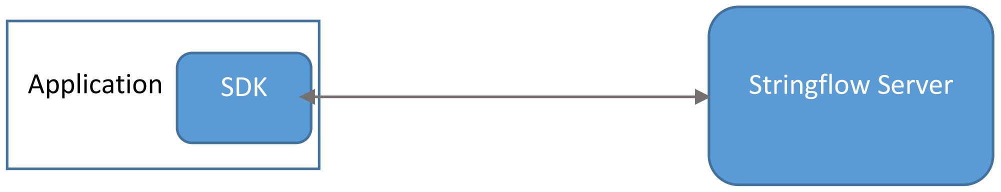
Overview

Stringflow Server Technology offers highly scalable messaging infrastructure with unprecedented administration control and out-of-the-box clustering capability making it highly available technology. Stringflow is an XMPP compliant server with multiple proprietary extensions such as Stringflow Content Model (SF Content Model) which is used to transmit media across network. As the server is XMPP compliant, it is possible to access Stringflow server with any XMPP client library of your choice, however the library can't take advantage of Stringflow proprietary extensions. Therefore, it is recommended to use Stringflow Client Library. Stringflow client libraries aka SDK(s) are available in following languages-

- Java
- Android (Java)
- iOS (Swift)
- Windows (C#)
- Java Script
- Python

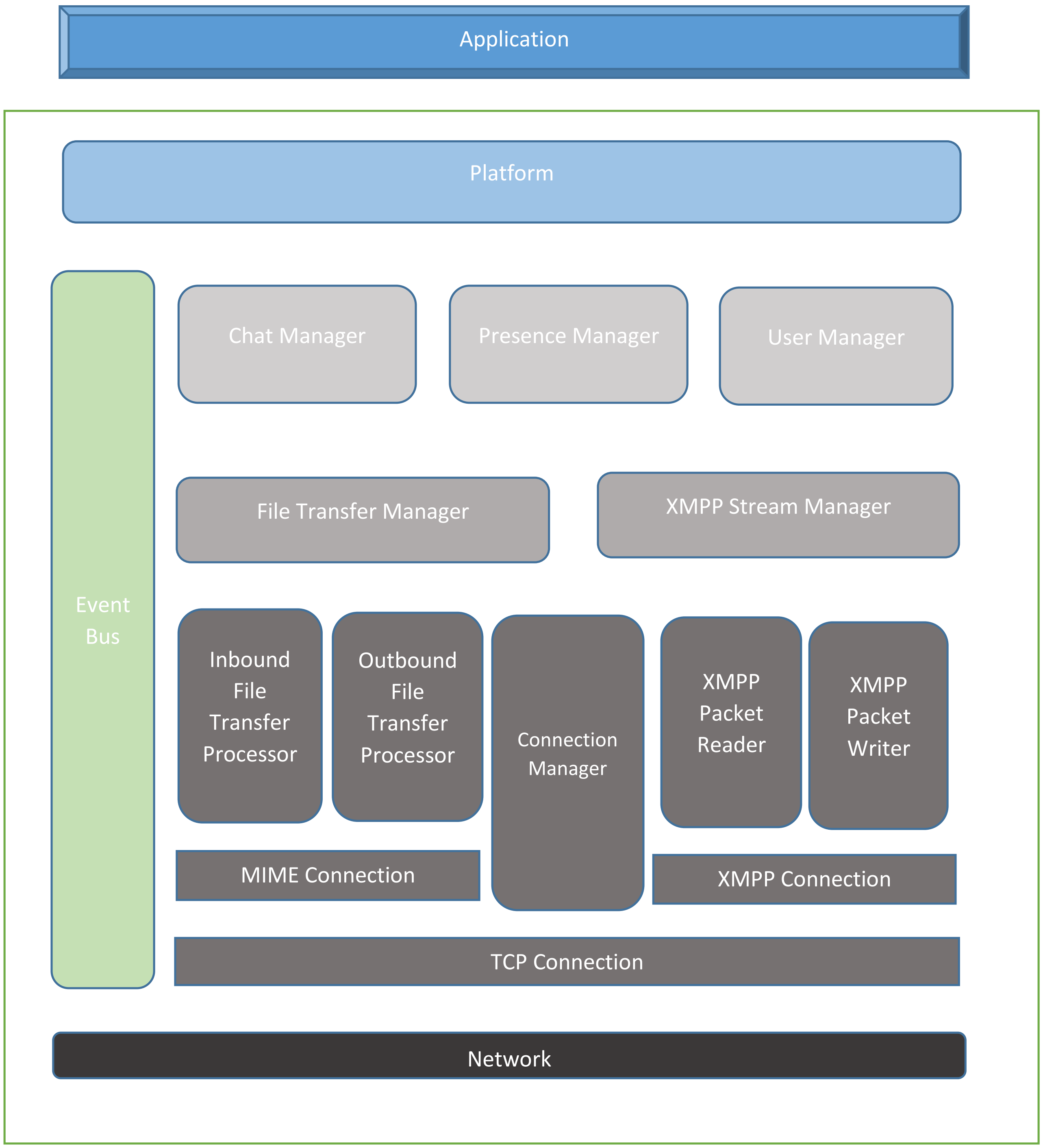
Unlike HTTP, XMPP is not-so-well-known protocol among developers; and the state-full nature of the XMPP protocol makes its semantics involved and vast. Long lived connections with fully duplex behavior adds up to situation. Therefore Stringflow SDK has been designed for seamless communication with Stringflow server with minimum effort from developer. There has been a well-thought architecture underneath making it robust and predictable at the same empowering developers to make maximum use of XMPP protocol semantics as well as Stringflow Server Technology. In order to shorten the learning curve, all the Stringflow SDK(s) follow the same architecture.

At a very high-level, the SDK works as shown in the below diagram where application communicates to the SDK which interns communicates to the server.



Other than connecting to Stringflow server, SDK(s) have various developer friendly constructs which allow developers to do many complex things with ease. Therefore it is important to grasp the core concepts of SDK architecture. For more documentation and samples, please refer to Stringflow github account under organization: ALTERBASICS (see appendix).

Below is a high-level architecture of Stringflow Core SDK-



As shown in the above diagram, Stringflow SDK has a layered architecture in which whole SDK has been divided into various layers communicating to each other; each layer consists of a set of entities.

Network Layer

XMPP protocol requires a long-lived persistent network connection (TCP Connection) which is full-duplex in nature. Network layer within SDK is responsible for maintaining the underlying TCP connection between SDK and server. *TCPConnection, XMPPConnection, ConnectionManager* and *ReconnectionManager* are part of network layer.

ConnectionManager is responsible for managing life-cycle of a network connections within SDK. By design, all the entities which require a network connection borrow it from *ConnectionManager.* As there is only one XMPP connection, the same instance is returned in the response of a borrow request. Due to restricted access, Network connections (*TCPConnection/XMPPConnection*) can't be instantiated outside of network Layer. Connection managers ensures that there is only ONE connection alive at any given point in time for XMPP stream packet exchange (although there may be other connections created to support Stringflow Content Model aka SFCM). It also ensures that connections are cleaned-up when application shuts down SDK.

With unstable network such as mobile devices, network connections are bound to break frequently. In such cases client needs to detect broken connections, re-establish the connection with server and ensure redelivery of the user packets which may have been written to the network socket buffer but couldn't be transmitted to server. In SDK, *ReconnectionManager* is responsible for making attempts to re-establish network connection with server. There are various Reconnection Policies within SDK which determines the frequency at which *ReconnectionManager* will attempt to reconnect with server.

IO Layer (Input-Output Layer)

IO Layer sits on the top of network layer and performs following tasks-

- Reading bytes from network
- Generating Packets
- Writing User Packets on to network
- Processing packet level acknowledgements

Reader, *Writer* and *XMPPStreamManager* are the major stakeholders in IO layer. *Reader* and *Writer* are continuously running threads sharing the same underlying connection for reading and writing bytes respectively. Reader continuously reads bytes from network, generates packets and forwards them to its subscribers (for more information please refer to Stringflow Collector/Forwarder framework). *XMPPStreamManager* is always subscribed to *Reader* for packet collection.

Similarly *Writer* receives all the packets which needs to be written on to the wire (connection). As the underlying connection is not always in CONNECTED state (subjected to network availability on device), these packets are queued up in *Writer* thread which are written to network as and when *Writer* thread can do so.

XMPPStreamManager is the entity which does the orchestration among *Reader*, *Writer* and *ConnectionManager*.

Application Layer

Application layer sits on top of the IO layer in SDK. It consists of the classes which are accessed by application to communicate with SDK. Following are the major entities which application developers are likely to use while integrating Stringflow SDK-

- Chat Manager
- Presence Manager
- User Manager

Each of these entities have clearly defined responsibilities; for example *UserManager* deals with managing user level transactions within SDK such as fetching user Roster from server, updating user roster, fetching groups that user is part of etc. Similarly, as the name suggests, *PresenceManager* deals with managing user presence and receiving presence broadcasts from server for other users. *ChatManager* has all the convenience methods to send/receive chat messages.

This document is written for core SDK which is not any platform specific. SDKs such as Android and iOS has extensions of the application layer classes which offers platform specific convenience methods.

Platform

Platform is the custodian for various entities in SDK. It keeps instances of all the application layer classes (*ChatManager*, *PresenceManager* and *UserManager*) along with user session data and Event Bus.

As soon as SDK is loaded (using *SDKLoader*) into memory, various instances are registered within Platform which application developers can access at will.

Event Bus

Stringflow SDK has a concept of events where events can be raised at any of layers within SDK. These events are pushed to Event Bus which maintains active subscribers/handlers for each of the event types. As soon as an event is placed onto Event Bus, it executes the registered handlers for the event. The events raised within core SDK are consistent across all the implementations (Java SDK, C# SDK, Android SDK etc).

Additionally, application developers can define custom events and handlers. These events can be generated at any place and time during execution and pushed on to the Event Bus allowing developers to leverage reactive programming model.

Collector Forwarder (ColFor) Framework

As stated before, all Stringflow SDK(s) follow the same design principles to be consistent across platforms. In all the design decisions, ease of use is non-negotiable feature. While SDK(s) employs many levels of abstractions to reduce the number of lines of code required to be written by a developer, it also offers out-of-the-box features which developer can leverage in various situations.

Collector Forwarder framework is one of the central design principles that allows developers to tap into packet flow at any point within SDK. A *forwarder* is an entity which forwards the packets that it receives to its subscribers; on the other side a *collector* collects (receives) packets from a *forwarder*. There are many classes marked as *Collector*/*Forwarder* within SDK.

The framework can be leveraged in many ways. Developer can write a *Collector* and subscribe to any of the Forwarders within SDK tapping into the packet flow at that forwarder.

Appendix

Stringflow github account (https://github.com/AlterBasics) has lot of other documentations and samples.

Following repositories are available on githib-

- A sample Android application which uses Stringflow Android SDK
  https://github.com/AlterBasics/BEACH_Android

- A sample iOS application which uses Stringflow iOS
  https://github.com/AlterBasics/BEACH_iOS

- A command line Java client which uses Stringflow Core-Java SDK
  https://github.com/AlterBasics/sf_clc_java

- iOS SDK artifacts, documentation and samples
  https://github.com/AlterBasics/sf_sdk_iOS

- Android SDK artifacts, documentation and samples
  https://github.com/AlterBasics/sf_sdk_android

- Java SDK artifacts, documentation and samples
  https://github.com/AlterBasics/sf_sdk_java