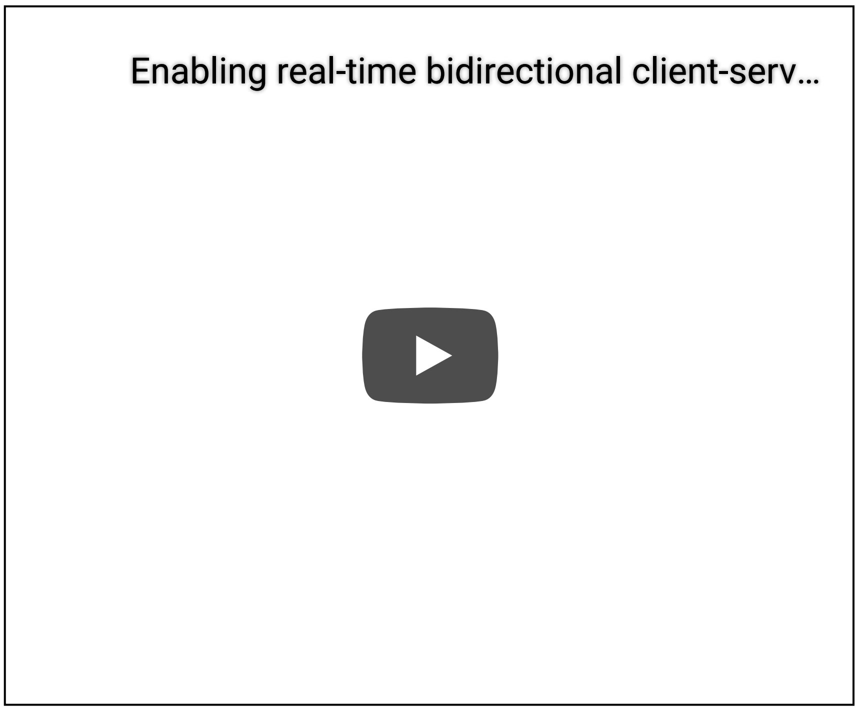
Tim Nusch
June 20, 2014 5 minute read

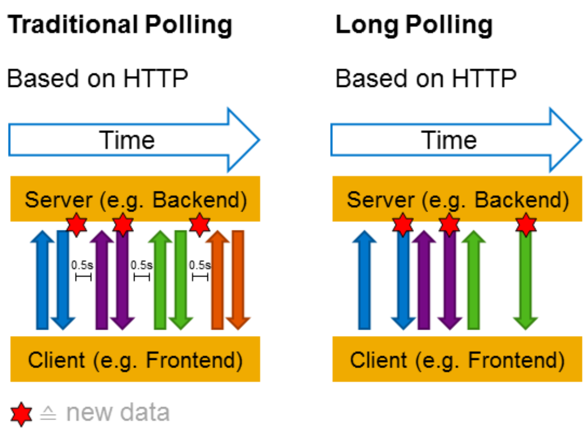
Enabling real-time bidirectional client-server communication using ABAP Push Channel

The client-server model is a typical architectural pattern in modern networks. But from an architect's/developer's point of view we are left with a full "zoo" of protocols and paradigms which can be used to enable communication between client and server. Unfortunately, a lot of commonly used techniques as polling are already prehistoric and their disadvantages outweigh their benefits.

Let's assume a SAP UI5-based chart rendered in the browser which gets fed with data coming from an ERP backend. If the backend data changes often and irregularly (stock prices, sales orders), the chart can become outdated very quickly and therefore needs to adapt to these changes in order to show the correct values. A demo video of the example can be viewed here or in SAP's media share at [1].

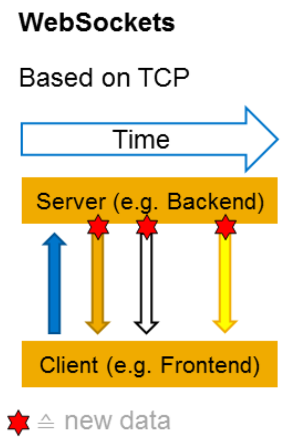


In the past we achieved this kind of client-server communication by polling techniques with multi-seconds intervals or just a refresh button to manually trigger an update. Usually we are using HTTP for the communication and open one connection for every update. The drawbacks of these approaches are lacks of built-in up-to-dateness, bad performance because of too many requests and a bad user experience because of unnecessary clicks.



This picture shows both traditional polling and an improvement called long polling. Both are based on HTTP and therefore always send complete request-response pairs. The red stars in the graphic show where new data gets available or data changes (new sales order, updated stock price).

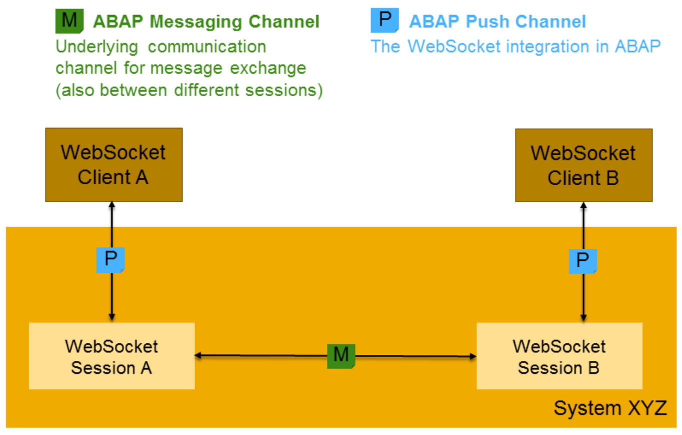
While using traditional polling e.g. with a time interval of 0.5 seconds, the frontend "asks" the backend in a frequent interval if new/updated data is available. The drawback of this approach is obviously that there is a lot of communication overhead since request-response-pairs are sent even if no data is available. Decreasing the overhead via increasing the polling interval leads to more misses and more outdated data on the frontend. While using long polling, the server only "answers" the requests when new data is available. While this approach performs better, especially when updates are non-frequent, there is still an unnecessary big overhead because plain old request-response-pairs are used.



HTML 5 WebSockets introduce a new way the client and the server communicate. WebSockets build a persistent connection between client and server by using only two HTTP messages; one handshake to establish and another to close the connection. The main difference to the polling approaches is that WebSocket messages are based directly on TCP (OSI Layer 4) while polling messages are based on HTTP (Layer 5-7). This eliminates the need to use HTTP request-response-pairs and opens the door to real-time communication from server to the client **and** from the client to the server.

The WebSocket protocol was standardized in 2011 by the IETF as RFC 6455 [2] and the WebSocket API is defined in a W3C specification [3]. WebSockets can use TCP port 80 which is usually used for HTTP connections. Because of that it can also be used with firewalls which block non-web Internet connections. The secure version of WebSockets (wss://) is supported by several modern browsers as Chrome 14 (released in 2011), Firefox 6 (2011), Safari 6 (2012) and IE 10 (2012).

But WebSockets also have some drawbacks: The application of the Same Origin Policy which restricts browser-side programming scripts in accessing content – e.g. the DOM – of pages with different origin is not required by the protocol (neither in [2] nor in [3]). However, developers and security consultants should always implement or try to make it a requirement to check the "Origin" header at the server to secure web applications from Cross-Site Hijacking attacks. Another disadvantage of WebSockets is that you are restricted to the basic datatypes BINARY and STRING, which means you have to care yourself about data conversion. Apart from that, it has to be kept in mind that older browsers don't support WebSockets and proxys have to support WebSockets as well.

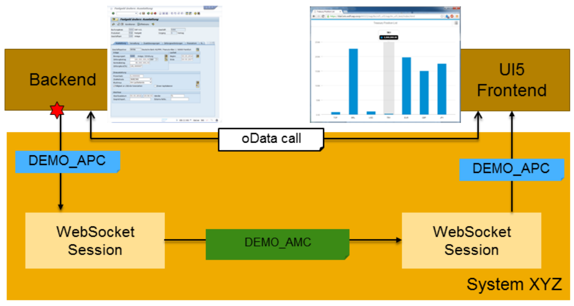


SAP's ABAP framework of WebSockets is called ABAP Push Channel (APC). APC is available as of SAP Netweaver AS ABAP 7.40 support package 5 (SP5) as part of the ABAP Channels infrastructure. Together with ABAP Messaging Channel (AMC), APC enables the development of interactive scenarios. The basic architecture as shown in picture 3 consists of several WebSocket clients which communicate via APCs – the ABAP implementation of the actual WebSocket messages – through AMC, which acts as an underlying communication channel. If you want to learn more about ABAP Channels architectures, please refer to Masouds [4] SCN article.

Let's assume we want to refresh an UI5 chart as soon as data updates on an ERP backend system.

Having that in mind, we can follow two different patterns to use APC. The first one is to send trigger messages from the backend, transporting only the information that something has changed without sending actual data. At frontend side we can react to that trigger message with a full update of the OData entity set. This first pattern has the advantage that it is very easy to implement and that the trigger messages are lean. The clear drawback is that full updates might not be needed in every scenario and there is an additional roundtrip (e.g. an OData refresh).

Apart from that we can send actual data within the messages for partial updates or recalculations. While this second pattern is more difficult to implement and it has to be kept in mind that only BINARY and STRING datatypes can be sent through APCs and therefore a data conversion / serialization might be needed, it improves the overall performance and the latency.



The exemplary architecture enhances the general ABAP Channel architecture basically at two points. The first one is the additional oData call from the UI5 frontend back to the ABAP backend (first pattern was used) and the second one is a tiny modification in the backend. Without going into detailed specifics, we need to set up the backend in that way that the APC message is sent automatically when data is saved / persisted. This could be achieved by adding the APC call into the V2 update task which is always executed after the DB commit. There exist some other ways to initiate the APC call but this highly depends on the scenario.

An exemplary code example for the UI5 frontend which reacts to APC message coming from the backend could look like asfollows:

```
// Open connection to WebSocket "demo_apc" on system "xyz"

ws = new WebSocket("wss://xyz.wdf.sap.corp:44322/sap/bc/apc/sap/demo_apc");


// React to incoming APC message (second pattern)

   a. ws.onmessage = function (evt) {

   var oModel = new
sap.ui.model.odata.ODataModel("/sap/opu/odata/sap/YOUR_DEMO_UI5_APPLICATION/");
sap.ui.getCore().setModel(oModel);
oChart.bindRows("/Values");
};
```

For further hands-on-tutorials and more code examples (both Javascript on the client side and ABAP on the server side) please refer to the following SCN article [5].

As a conclusion, it has been shown that there is a possibility to enable real-time bidirectional client-server communication via APC, the ABAP implementation of WebSockets. But as discussed please keep also in mind that APC is no magic bullet for all client-server scenarios.

[1] https://mediashare.wdf.sap.corp/public/id/0_1f1wupq3 (only SAP internally)

[2] http://tools.ietf.org/html/rfc6455

[3] http://dev.w3.org/html5/WebSockets/

[4] http://scn.sap.com/community/abap/connectivity/blog/2013/11/18/websocket-communication-using-abap-push-channels

[5] http://scn.sap.com/docs/DOC-53598

Assigned tags

~~Like~~ ~~Alert Moderator~~

ABAP Connectivity | Mobile | abap | abap push channel | channel |

View more...

Related Blogs

Real-time notifications and workflow using ABAP Push Channels (websockets) Part 1: Creating the APC and AMC
By **Brad Pokroy** , Oct 16, 2014

Get your hands on Industrial IoT scenarios with ABAP Channels
By **Olga Dolinskaja** , Feb 22, 2017

Introduction to ABAP Channels
By **Olga Dolinskaja** , Nov 27, 2014

Related Questions

ABAP Push Channel is not supported in the selected project
By **Former Member** , Nov 03, 2016

Proxy call from ECC to PI or Vice versa
By **Abhishek Gupta** , Apr 25, 2017

ABAP Push Channel ...How to deal with data?
By **Amol Samte** , Oct 21, 2016

2 Comments

You must be Logged on to comment or reply to a post.

Brad Pokroy

August 18, 2014 at 4:58 am

Hi Tim,

Great blog!

In the first pattern where you refer to sending a trigger message to start a new oData call, you mention a drawback being that full updates may not be needed. There is a way around this, if your oData service has implemented delta query support then the updates would be more efficient and there would not be a need do a full query each time.

Cheers,

Brad

Like (0)

Tim Nusch | Post author

August 18, 2014 at 6:57 am

Hi Brad,

I wasn’t aware of that! I will update the blog accordingly. Thank you!

Cheers,

Tim

Like (0)

Share & Follow