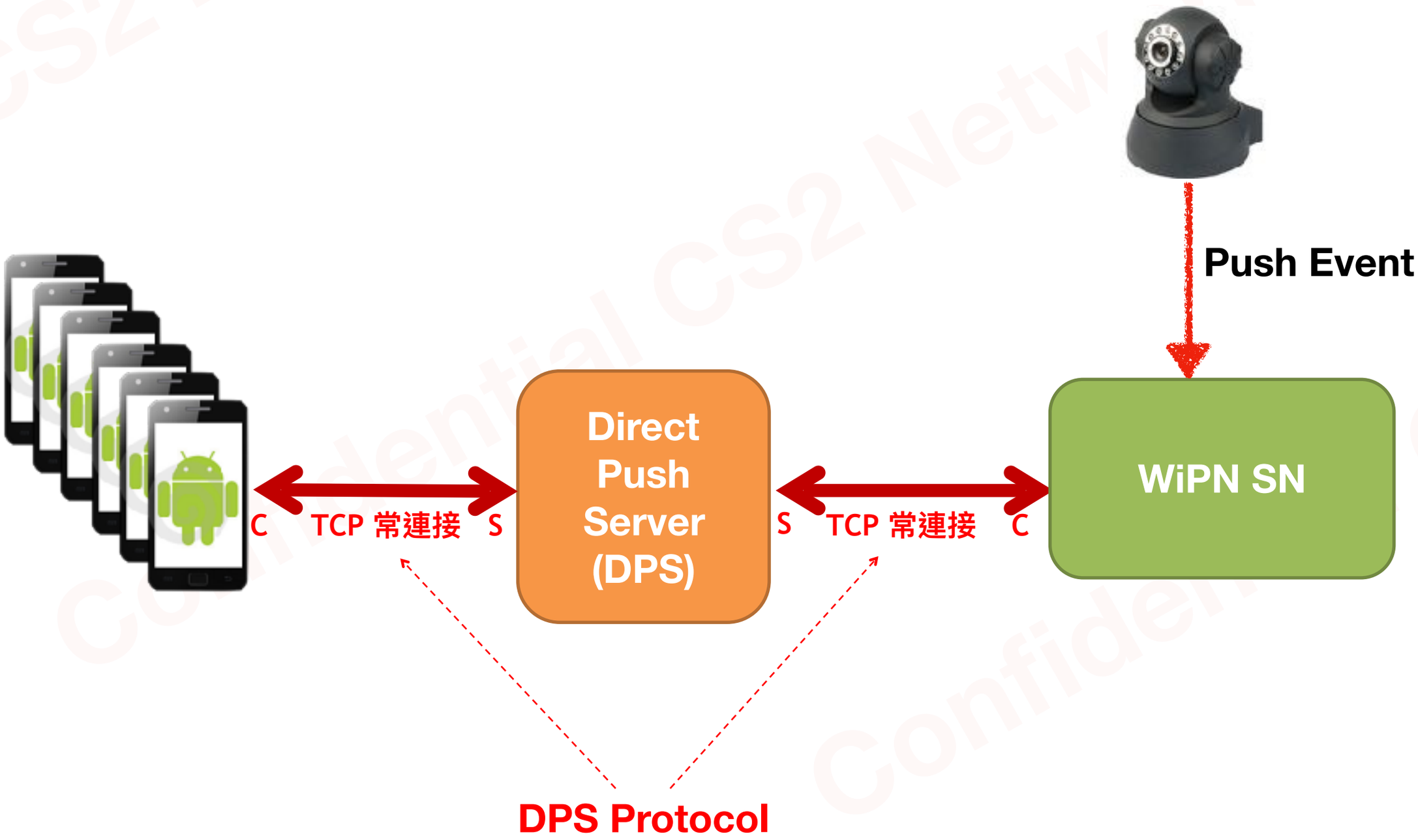
DPS
WiPN Direct Push Service

CS2 Network
Charlie

Date: 2018/12/01
Doc. Ver: 0.1

Direct Push Architecture



DPS Token

- Every App need a Unique DPS Token before making Subscription
  - DPS Token is used to distinguish every App
- DPS Token is controlled by DPS Server
  - App send Token request to DPS Server
  - App must save the Token
- Every 5 minute App send Login to DPS Server.
  - App need to provide its DPS Token when Login
- DPS Token is a 32 Byte Hex Number (0~F)
  - For example: ‘57FA5CC000000009ED9DB4CDA51D6907’

DPS API

- CHAR* DPS_GetAPIVersion(UINT32 *Version);
- INT32 DPS_Initialize(const CHAR *ServerIP, const UINT16 ServerPort, const CHAR *AES128Key, const UINT16 PortNo);
- INT32 DPS_DeInitialize();
- INT32 DPS_TokenAcquire(CHAR* TokenBuf, const UINT16 Size);
- INT32 DPS_RecvNotify(const CHAR *Token, CHAR *NotifyContent, UINT16 *Size, UINT32 TimeOut_ms);
- INT32 DPS_GetLastAliveTime(UINT32 *Time_Sec);

DPS API Error Code

ErrCode	Number	Description
ERROR_DPS_Successful	0	API is successfully executed
ERROR_DPS_NotInitialized	-1	DPS_Inistialize is not called yet
ERROR_DPS_AlreadyInitialized	-2	DPS_Inistialize is called already
ERROR_DPS_TimeOut	-3	Time out
ERROR_DPS_FailedToResolveHostName	-4	Can’t resolve Server Name
ERROR_DPS_FailedToCreateSocket	-5	Socket create failed
ERROR_DPS_FailedToBindPort	-6	Socket bind failed
ERROR_DPS_FailedToConnectServer	-7	Connection to DPS Server failed
ERROR_DPS_FailedToRecvData	-8	Failed to Receive data from Server
ERROR_DPS_NotEnoughBufferSize	-9	Buffer size is not enough
ERROR_DPS_InvalidAES128Key	-10	AES128Key string must exactly 16 Byte
ERROR_DPS_InvalidToken	-11	The Token string is not valid
ERROR_DPS_OnRecvNotify	-12	You can’t call this function while DPS_RecvNotify() is running
ERROR_DPS_OnAcquireToken	-13	You can’t call this function while DPS_TokenAcquire() is running
ERROR_DPS_NotOnRecvNotify	-14	You can’t call this function while DPS_RecvNotify() is not running

DPS API Version

- Function Declare:
  - **CHAR* DPS_GetAPIVersion(UINT32 *Version)**
- Description:
  - **DPS_GetAPIVersion: To retrive DPS API version information and library descriptions.**

    This function is always executable and will successfully return correct version info and library description.
- Parameters:
  - **Version: The pointer to a unsigned integer version number. 0x01020304 → Version: 1.2.3.4**
- Return:
  - The description string of this DPS library.
- Uage example:
    - *printf("%s\n", DPS_GetAPIVersion(&Version));*

Initialize / DeInitialize

- Function Declare:
  - **INT32 DPS_Initialize(const CHAR *ServerIP, const UINT16 ServerPort, const CHAR *AES128Key, const UINT16 PortNo)**
  - **INT32 DPS_DeInitialize()**
- Description:
  - **DPS_Initialize**: To initialize usage of DPS library.
  - **DPS_DeInitialize**: To free all resource used by DPS library.
- Parameters:
  - **ServerIP**: The IP or Host name of DPS Server.
  - **ServerPort**: The service port number of DPS Server.
  - **AES128Key** : The encryption key string. This shall be the same as which set in DPS Server. (Please Keep AES128Key as top secrect)
  - **PortNo**: The local port number which DPS library will bind to. If '0' is specified, random port number is used.
- Return:
  - ERROR_DPS_AlreadyInitialized
  - ERROR_DPS_InvalidAES128Key
  - ERROR_DPS_FailedToResolveHostName
  - ERROR_DPS_FailedToBindPort
  - ERROR_DPS_FailedToCreateSocket
  - ERROR_DPS_FailedToConnectServer
  - ERROR_DPS_Successful

Token Acquire

- Function Declare:
  - **INT32 DPS_TokenAcquire(CHAR* TokenBuf, const UINT16 Size)**
- Description:
  - **DPS_TokenAcquire**: To acquire a new token from DPS. A token is a 32 Byte string, which contains '0~9', and 'A'~'F' only. Token is like an unique id so that DPS can tell every Client. Token will be needed in DPS_RecvNotify(). **Note: You just need to acquire token once, and then keep it in disk space or flash.**
- Parameters:
  - **TokenBuf**: the string buffer to receive Token from DPS server.
  - **Size** : The size of TokenBuf. **(Must be larger than 32 byte!)**
- Return:
  - ERROR_DPS_NotInitialized
  - ERROR_DPS_OnRecvNotify
  - ERROR_DPS_NotEnoughBufferSize
  - ERROR_DPS_OnAcquireToken
  - ERROR_DPS_FailedToBindPort
  - ERROR_DPS_FailedToCreateSocket
  - ERROR_DPS_FailedToConnectServer
  - ERROR_DPS_FailedToRecvData
  - ERROR_DPS_Successful

Wait and Receive Notify

- Function Declare:
  - **INT32 DPS_RecvNotify(const CHAR *Token, CHAR *NotifyContent, UINT16 *Size, UINT32 TimeOut_ms)**
- Description:
  - **DPS_RecvNotify**: This function is used to wait and receive notification from DPS server. **DPS_RecvNotify()** will block until notification arrived or timeout. If a notification is successfully received, ***NotifyContent** will contains notifying contents and ***Size** is its size. If ***Size** is smaller than content received, contents will be truncated. To prevent truncation of receiving notification contents, the **NotifyContent** should be at least 1440 Bye.
- Parameters:
  - **Token**: the Token string got from **DPS_TokenAcquire()** previously.
  - **NotifyContent**: The Buffer used to receive Notifying contents. Suggested size is 1440 Byte.
  - **Size**: When calling, ***Size** is telling max size of **NotifyContent**. When returned ERROR_DPS_Successful, ***Size** is telling effective data size of **NotifyContent.** It is suggested that size of **NotifyContent** shall be 1440 Byte.
  - **TimeOut**: Specifying the maximum waiting time in ms. If 0 is specified, **DPS_RecvNotify()** will block indefinitely.
- Return:
  - ERROR_DPS_NotInitialized
  - ERROR_DPS_OnAcquireToken
  - ERROR_DPS_OnRecvNotify
  - ERROR_DPS_InvalidToken
  - ERROR_DPS_FailedToBindPort
  - ERROR_DPS_FailedToCreateSocket
  - ERROR_DPS_FailedToConnectServer
  - ERROR_DPS_FailedToRecvData
  - ERROR_DPS_InvalidToken
  - ERROR_DPS_Successful

Check Connection to DPS Server

- Function Declare:
  - **INT32 DPS_GetLastAliveTime(UINT32 *Time_Sec)**
- Description:
  - **DPS_GetLastAliveTime**: This function is used to check if connection to DPS Server is ok or not.
    - When **DPS_RecvNotify()** is called, the DPS API will establish a connection to DPS Server, so that DPS Server may send Notify via this connection.
    - This **DPS_GetLastAliveTime()** is used to retrive the time in seconds before, when the last heart-bit from DPS Server arrived.
    - Due to possibility of various network loss problems, you shall periodically call **DPS_GetLastAliveTime()** every 30~300 second (depends how real time your application require)
    - If you found the **Time_Sec** returned is larger then **40**, and **DPS_RecvNotify()** is still blocking you shall call DPS_DeInitialize(), then DPS_Initialize(), and then **DPS_RecvNotify()** again**.**
- Parameters:
  - **Time_Sec**: If Successful, **Time_Sec** tells when is the last hear-bit from DPS Server received **in unit of second before.**
- Return:
  - ERROR_DPS_NotInitialized
  - ERROR_DPS_Successful

Working Flow of DPS APP