Stack SMS

Chema Alonso Pablo González Lucas Fernández Francisco Ramírez

CDO, Telefónica
Madrid, Spain

chema@11paths.com
pablo.gonzalez@11paths.com
lucas.fernandezaragon@telefonica.com
franciscojose.ramirezvicente@telefonica.com

Abstract- El envío de mensajes de texto a través del protocolo *SMS* sigue siendo muy popular a día hoy. En muchas regiones, incluso enviar un mensaje de texto es totalmente gratis, siendo por lo tanto ampliamente utilizado, ya que no requiere de acceso o conexión a Internet para su funcionamiento. Esto es una ventaja, ya que abre una vía de comunicación para situaciones donde no se encuentre disponible una conexión de datos. Uno de los casos prácticos que podemos imaginar es el control de dispositivos de *IoT* que no tengan conexión a Internet, pero sí cobertura móvil. *Stack SMS* (arquitectura y *SDK* bajo licencia *Open Source*) pretende aprovechar sobre todo de esta última ventaja, la opción de tener acceso a un canal de comunicaciones usando simplemente una conexión *GSM*. Pero, además, el diseño a modo de protocolo de comunicaciones permite implementar otro tipo de aplicaciones más complejas siempre usando como base el envío de mensajes tipo SMS. También ofrece una librería o *framework* para programar todo tipo de utilidades o aplicaciones que puedan ser ejecutadas dentro de *Stack SMS*.

1. INTRODUCCIÓN

SMS o *Servicio de Mensajes Cortos* (en inglés, *Short Message Service*) [1] es un servicio que lleva operativo desde 1985 sin sufrir grandes cambios en su formato hoy en día. De hecho, sigue funcionando incluso en las nuevas redes 4G y 5G enviando mensajes a través del estándar *GSM*. Estos mensajes tienen un tamaño pequeño, unos 140 caracteres, se envían básicamente para enviar mensajes de texto en diferentes teléfonos móviles. Una de sus grandes ventajas, la cual le ha permitido sobrevivir hasta hoy día, es su capacidad de envío a través de redes con únicamente cobertura *GSM*. Esta característica hace que recibir o enviar un *SMS* sea la única forma de enviar información sin realizar una llamada cuando no existe una cobertura de datos.

Pero, además, en el encapsulamiento de estos *SMS* también es posible introducir otro tipo de información, es decir, cualquier tipo de datos. De esta forma sería factible diseñar una aplicación cliente la cual fuera capaz de recibir estos *SMS* con otro tipo de información encapsulada y de esta forma permitir realizar todo tipo de operaciones sobre el dispositivo que ha recibido los mensajes. Es decir, codificar una trama con diferentes cabeceras de información la cual sería procesada en el otro extremo para realizar todo tipo de operaciones. Se puede entender como una pila de protocolos creando un protocolo de comunicaciones sobre *SMS*. De esta forma es posible añadir este tipo de características, como por ejemplo enviar un fichero, en entornos donde sólo exista cobertura *GSM*. Uno de esos escenarios podría ser una catástrofe natural en la cual no exista servicio de Internet móvil, pero sí cobertura telefónica.

Los casos de uso donde este conjunto de protocolos o pila podría ser utilizado es muy amplio. De hecho, es totalmente amoldable a las necesidades del usuario que necesite implementar un protocolo superior para cualquier tipo de aplicación que desee. Otro de los campos en el cual se abren muchas posibilidades en el mundo del *IoT* al permitir encapsular en *SMS* opciones de ejecución más complejas para diferentes dispositivos, así como ofrecer una forma de control cuando no existe conectividad a Internet, o la calidad de señal es baja, pero sí *GSM*.

1.1. Arquitectura protocolo *SMS* estándar

Como se ha comentado anteriormente, los mensajes *SMS* están limitados a 120 bytes (en España tienen un tamaño variable), pero existen varios tipos de set de caracteres que se pueden utilizar según su codificación. Por ejemplo, si se pueden codificar como 140 caracteres con 8 bit o 70 caracteres usando 16 bits. Pero hay que destacar que todos los sistemas *GSM* soportan el modo principal de codificación, *GSM-7 bit [2]*.

	0	1	2	3	4	5	6	7
0	@	Δ	SP	0	¡	P	¿	p
1	£	_	!	1	A	Q	a	q
2	$	Φ	"	2	B	R	b	r
3	¥	Γ	#	3	C	S	c	s
4	è	Λ	¤	4	D	T	d	t
5	é	Ω	%	5	E	U	e	u
6	ù	Π	&	6	F	V	f	v
7	ì	Ψ	'	7	G	W	g	w
8	ò	Σ	(	8	H	X	h	x
9	Ç	Θ	)	9	I	Y	i	y
10	LF	Ξ	*	:	J	Z	j	z
11	Ø	Note 1	+	;	K	Ä	k	ä
12	ø	Æ	,	<	L	Ö	l	ö
13	CR	æ	-	=	M	Ñ	m	ñ
14	Å	ß	.	>	N	Ü	n	ü
15	å	É	/	?	O	§	o	à

Figura 1: Alfabeto GSM 7 bit ASCII

Un mensaje *SMS* tiene, de forma general, el siguiente formato de paquetes:

- Longitud del *SMSC* Información del servicio o proveedor de comunicaciones que emite el *SMS*
- *Timestamp SMSC* Datos de tiempo para sincronizar el envío del *SMS* emitido por el *SMSC*
- Número de teléfono del emisor Número de teléfono desde el cual se envía el mensaje
- Identificador de protocolo *TP-PID* identificador del protocolo utilizado
- Codificación de los datos *TP-DCS* tipo de codificación utilizada en el *SMS*
- Longitud del mensaje enviado *TP-UDL* longitud del mensaje
- Datos de usuario (texto del mensaje) *TP-UD* datos enviados por el usuario (texto del *SMS*)

Stack SMS aprovecha toda la infraestructura *SMS* y encapsula el protocolo de envío dentro de "Datos de usuario" (*TP-UD*) en formato texto base64 (cabecera en hexadecimal, cuerpo en base64/ cifrado). Por lo tanto, los usuarios recibirán un *SMS* tradicional, siendo la aplicación cliente la que se encargue de extraer, comprobar y ejecutar las instrucciones encapsuladas *Stack SMS* dentro de dicho mensaje de texto.

2. STACK SMS

La pila o *Stack SMS* presentado en este documento tiene como función exactamente implementar un conjunto de protocolos y un *SDK* (para facilitar su implementación), orientado a las conexiones, el cual sea capaz de encapsular a su vez protocolos superiores. De esta forma sería posible implementar todo un canal de comunicaciones de datos a través de este protocolo.

2.1. Arquitectura *Stack SMS*

En este apartado se muestra la composición del protocolo inferior del *Stack SMS*. Este protocolo tiene funciones similares a las de *TCP* en la arquitectura *TCP/IP [3]*. A continuación, se muestra el formato de trama y la descripción de los campos que constituyen el protocolo.

2.2.1. *Formato de trama*

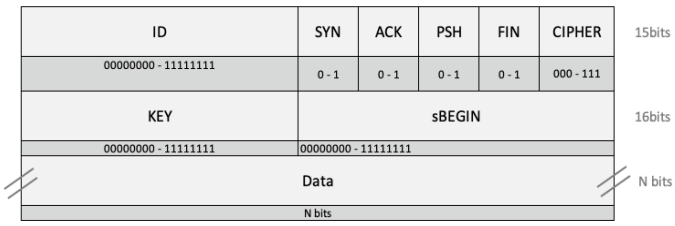

Figura 2: Formato de trama Stack SMS

ID (8 bits): identifica el tipo de protocolo utilizado en el encapsulamiento de la información, es decir, en el campo de datos (*Data*).

SYN – ACK – PSH – FIN (1 bit cada uno): flags para el control de la conexión y trazabilidad de la información. Si la información es mayor de 140 caracteres, esta se divide en varios *SMS*. Aquí es donde entran en función estas *flags* las cuales permiten sincronizar y reunir la información en caso de estar segmentada. A continuación, se detalle el uso de cada *flag*:

- *SYN*. Este *flag*, tal y como ocurre en el protocolo TCP, indica un inicio de conexión contra el otro dispositivo a través del envío de un *SMS*.
- *ACK*. Este *flag* indica que se han recibido datos. Funciona igual que un asentimiento en *TCP/IP*.
- *PSH*. Este *flag* indica que se están enviado datos en el campo "data".
- *FIN*. Este *flag* indica que se quiere finalizar la conexión.

Cabe destacar que *Stack SMS* permite optimizar el inicio de una conexión y la finalización de ésta bajo el mismo *SMS*. Se puede encontrar una trama que tenga *SYN*, *PSH* y *FIN* activos, con lo que se puede iniciar una nueva conexión, enviar datos y cerrar la conexión en el mismo mensaje. Esto puede ser utilizado cuando se envía una cantidad pequeña de información.

CIPHER (3 bits): Este campo indica el cifrado que se aplica a la parte de datos encapsulados en el protocolo. Los tipos son: 0 (000) para *Base64*, 1 (001) para *AES+Base64* y 2 (010) para clave pública. Esto puede ampliarse en el futuro con nuevas implementaciones. El campo queda abierto y se ha reservado 3 bits para aplicar otros tipos de cifrado.

KEY (8 bits): en este campo se almacenará el *id* interno de *Stack SMS* para identificar el mensaje, necesario para tareas *posteriores* de sincronización y recuperación. Cuando una conexión comienza se genera una clave (*key*) que es enviada. De esta forma el receptor sabe la conexión que está abierta y puede identificar cuando un paquete de datos con el *flag PSH* es para una aplicación concreta o para otra. Por cada conexión nueva se genera una conexión. Esto funciona de forma análoga a los puertos en el protocolo *TCP*.

sBEGIN (8 bits): posición del mensaje, necesario para la trazabilidad en caso de enviar varios mensajes *SMS*. Este campo va ligado al campo *Key*. Para una misma *Key* podemos ir recibiendo diferentes datos encapsulados y podemos conocer la posición de esos datos gracias a este campo.

Data (N bits): contenido de los datos enviados. El tamaño varía en función del país.

Una vez se forma la cabecera, todo se codifica en hexadecimal para ahorrar espacio y de esa forma reducir el número de mensajes *SMS* a enviar.

2.2.2. Formato de conexión

Es formato de conexión de *Stack SMS* intenta ser lo más sencillo posible actuando de capa de transporte para cualquier otro servicio que se implemente en este sistema (se muestran algunos ejemplos más adelante). El esquema de conexión cuando un nuevo mensaje es creado (*new message*) se muestra en la figura 3. En él podemos apreciar la preparación de la información a enviar indicando entre otros campos, el tipo de cifrado (*CIPHER*) o qué porción del mensaje se está enviando (*sBEGIN*).

El segundo esquema, detalla las conexiones realizadas si ocurre algún error en el mismo proceso de creación de un nuevo mensaje (*new message error*), figura 4. Podemos apreciar que dichos mensajes mantienen toda la información de la conexión para poder conseguir una trazabilidad y de esa forma gestionar el error.

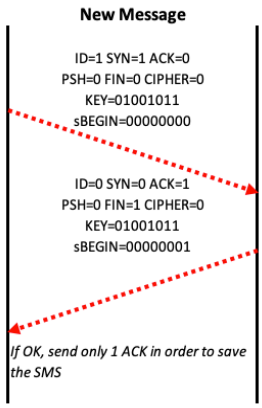

Figura 3: Formato de conexión para nuevo mensaje (inicio de la comunicación)

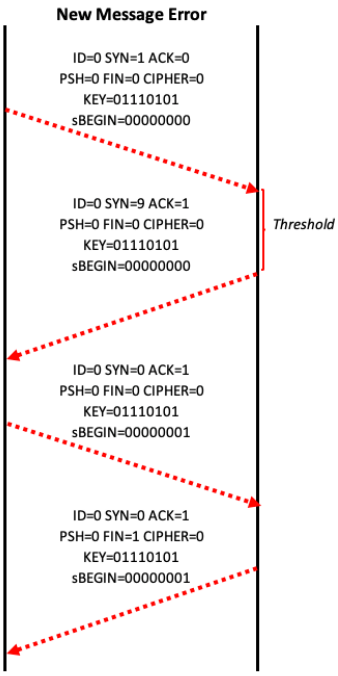

Figura 4. Formato de conexión correspondiente a un error a la hora de configurar la conexión

2.2.3. Codificación de los campos

Como se ha indicado anteriormente, la información *Stack SMS* se codifica en *base64*. En cambio, las cabeceras tienen un tratamiento especial. El método utilizado para la codificación de las cabeceras es a través de un *bitmask* (máscara de bits). La cabecera, al ser de tamaño fijo, se aplica un *padding* (añadir bits adicionales) por cada uno de los campos cuando se necesario (por ejemplo, si necesitamos completar el tamaño estándar de la cabecera). Una vez han sido codificados todos los campos, estos se concatenan y se codifican esta vez en hexadecimal.

2.2.4. Cifrado y codificación

Stack SMS ofrece de momento una opción de codificación (*Base64*) y otra de cifrado (*AES*) [4] las cuales pasamos a detallar:

- *Base64*: esta codificación se aplica al cuerpo del mensaje a transmitir. Durante el proceso de envío, *Base64* convierte la información en caracteres imprimibles del código *ASCII* (básicamente y en función de las variantes siempre son de A-Z, a-z y 0-9) ofreciendo simplificación a la hora de su manipulación y envío. Este mensaje se divide en varias n partes en función de su longitud, se codifica en *Base64* y finalmente se concatena con el resto de las cabeceras. En proceso inverso, de recepción, los mensajes se ordenan, se retiran las cabeceras, se une toda la información y finalmente se procede a su descifrado.

- *AES*: se nuevo se obtiene el contenido del cuerpo del mensaje principal, pero esta vez de comparte un secreto utilizando un *challenge* (se debe de responder correctamente con un "*response*" para proceder a su validación). El siguiente paso, una vez obtenida el secreto será cifrar el mensaje principal y dividirlo en varias *n* partes dependiendo de la longitud de este. Una vez recibido el mensaje ya procesado se envía y una vez recibido se ordena, se eliminan las cabeceras, se unen las diferentes partes y finalmente se procede a su descifrado.

Como nota final, hay que destacar que es posible puede enviar el contenido sin cifrar si así se requiere. El protocolo base del *Stack SMS* es flexible a este tipo de configuraciones.

3. PROTOCOLOS SUPERIORES

Una de las características más interesantes de *Stack SMS* y que ofrece gran potencial es la posibilidad de emular protocolos superiores de aplicación. De esta forma se le ofrece al usuario la creación de sus propios protocolos los cuales posteriormente puede encapsular dentro de *Stack SMS*. Vamos a ver a continuación algunos casos de uso:

3.1. Latch [5]

Latch es una aplicación y ofrece a su vez una colección de *SDKs* disponibles para *iOS* y *Android*, la cual permite añadir una capa adicional de seguridad a diferentes servicios. Es una herramienta muy intuitiva que permite, tal y como su propio nombre indica, utilizar un cerrojo virtual para bloquear o autorizar cualquier tipo de operación que tengamos configurada con *Latch*.

Stack SMS permite una integración con *Latch*, permitiendo usarlo como encapsulamiento para las comunicaciones entre *Latch* y la aplicación que tengamos protegida por este método.

Cabecera:

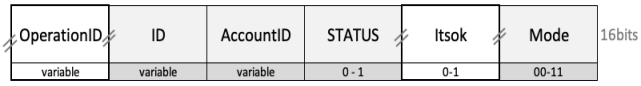

Figura 5. Cabecera para la integración de Latch en Stack SMS.

Los campos de la cabecera tienen la siguiente función:

OperationID (opcional, tamaño variable): Es el código para la operación que *Latch* va a consultar o modificar. Se relaciona con los 'cerrojos' asociados a una *App* en *Latch*.

ID (tamaño variable): identificador del paquete.

AccountID: es el código que identifica a un cerrojo principal de una app de Latch. Es un campo obligatorio. Se puede consultar o modificar el estado, en función del valor del campo *Mode*.

STATUS (1 bit): indica el estado de *Latch*, abierto o cerrado.

Itsok (1 bit): comprobación de realización de la operación.

Mode (2 bit): 01 para aplicar el estado de *Latch* y 10 para sólo consultarlo.

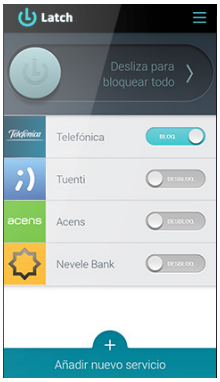

Figura 6. Ejemplo del panel de control Latch.

Conexión:

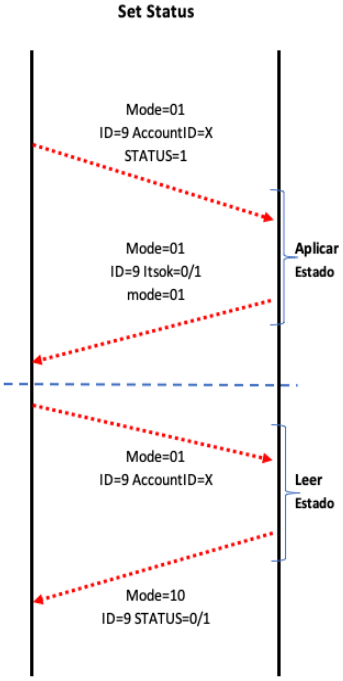

Figura 7. Formato de conexión para aplicar un estado Latch y consultarlo

3.2. Chat

Este otro ejemplo muestra como *Stack SMS* puede implementar una aplicación simple de *chat* ofreciendo nuevas características como la creación de grupos de envío, entre otras. La diferencia de este chat respecto a otros es que puede funcionar sin conexión a Internet, simplemente será necesaria una conexión *GSM*.

Cabecera:

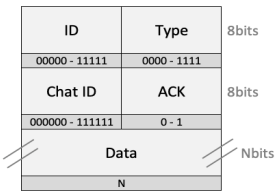

Figura 8. Cabecera para la implementación de un chat con Stack SMS.

ID (5 bits): Identifica el paquete

Type (4 bits): Sirve para identificar el tipo de operación a realizar en el chat (nuevo grupo, nuevo mensaje, etc).

ChatID (6 bits): identificador único del chat

ACK (1 bit): señal de confirmación de recepción.

Data (N bits): texto a enviar en el chat.

Conexión:

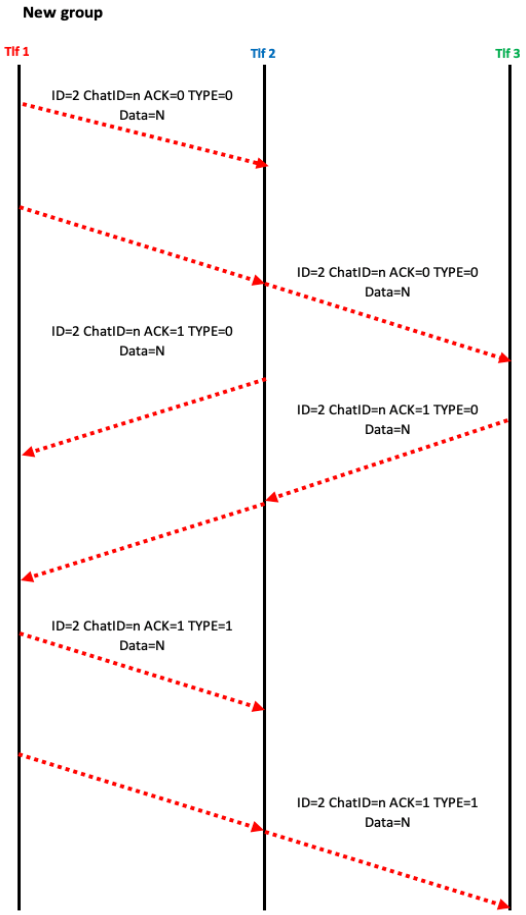

Figura 9. Formato de conexión para crear un nuevo grupo del chat con Stack SMS.

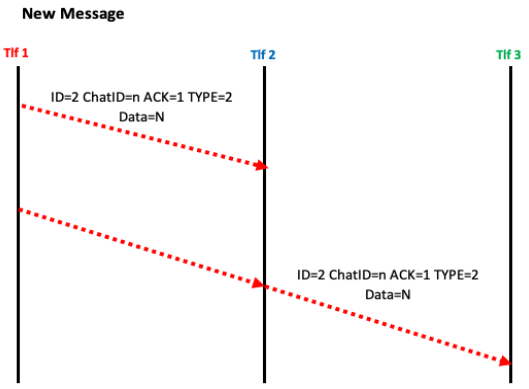

Figura 10. Formato de conexión para crear un nuevo grupo de chat con Stack SMS.

3.3. Juego adivina la palabra

El ejemplo siguiente muestra una posible forma de implementar una variante del famoso *Juego del Ahorcado [6]*. La finalidad es adivinar una palabra preguntando por una letra concreta en cada turno. Por cada letra que no se encuentre dentro de la palabra a adivinar, se restará una vida (equivalente al número máximo de intentos). El juego termina cuando se adivina la palabra dentro de los turnos establecidos o cuando ya no hay disponibles más vidas.

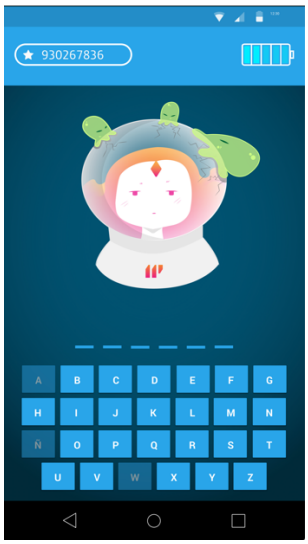

Figura 11. Implementación del juego (usando Stack SMS) advina la palabra creado por ElevenPaths.

Más allá de la intención específica de implementar este juego de adivinar la palabra, el objetivo final es demostrar la gran variedad de aplicaciones que se pueden crear, simplemente diseñando las tramas de los paquetes enviados *ad-hoc* según nuestras necesidades.

Cabecera:

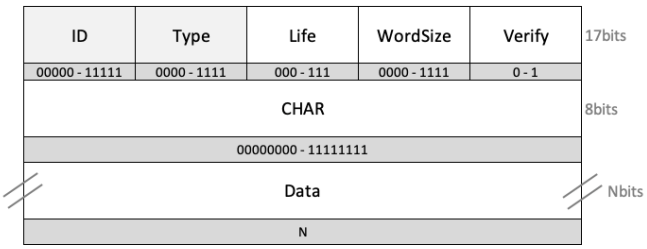

Figura 12. Cabecera juego ahorcado

ID (5 bits): identificador del paquete

Type (4 bits): tipo 0 indica nuevo juego y tipo 1 se utiliza para enviar una nueva letra para comprobar.

Life (opcional, 3 bits): indicará el número máximo de vidas disponibles (intentos)

Wordsize (opcional, 4 bits): asigna el máximo número de letras que componen la palabra

Verify (opcional, 1 bit): 0 indica partida perdida y 1 indicará partida ganada.

CHAR (8 bits): contiene el carácter en código ASCII que se utiliza en el juego para adivinar si existe en la palabra a descubrir.

Data (opcional, tamaño N variable): en campo almacenará la palabra completa que hay que adivinar.

Conexión:

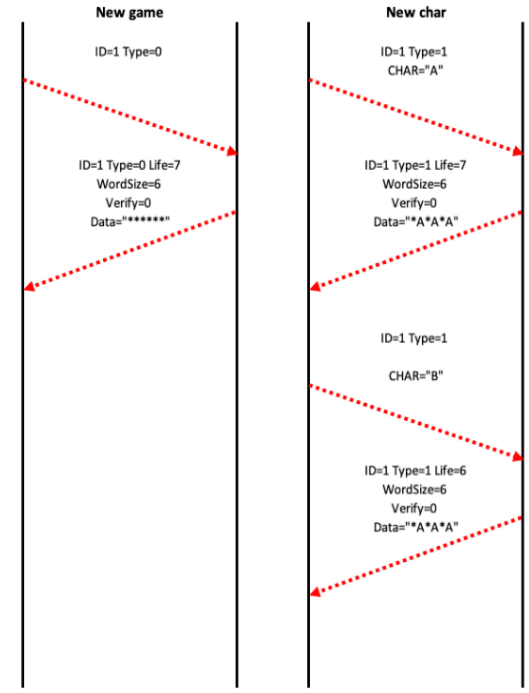

Figura 13. Formato de conexión para nuevo juego y envío de una letra durante el juego

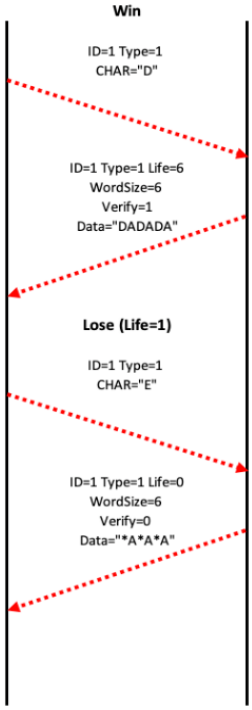

Figura 14. Formato de conexión para nuevo juego y nueva letra

3.4. Control GPIO Raspberry Pi [7] con Stack SMS (IoT)

Esta aplicación de *Stack SMS* se orienta más al *IoT*, utilizando para la implementación una *Rasbperry Pi* y el control de los puertos *GPIO* (*General Purpose Input Output)*. Utilizando *Stack SMS* podemos leer y escribir los datos de los puertos *GPIO* de dicho dispositivo. De esta forma, es posible leer datos por ejemplo de sensores, enviar datos a relés, motores, etc simplemente utilizando como canal de comunicación el *SMS* y *Stack SMS*. Existen muchos *shields* (placas de expansión) para *Raspberry Pi* que ofrecen conectividad *GSM*.

Cabecera:

ID	RaspModel	GPIOpinID	GPIOType	Mode	17bits
00000 - 11111	000 - 111	000000 - 111111	00 - 11	0 - 1	
GPIOData					Nbits
N					

Figura 15. Formato de conexión para nuevo juego y nueva letra

ID (5 bits): identificador del paquete

RaspModel (3 bits): modelo de *Raspberry Pi* para identificar el número de puertos *GPIO*. Los modelos A y B (los originales) (código 000) tienen 26 pines *GPIO*. Los modelos A+, B+ y B2 tienen 40 pines *GPIO* (001).

GPIOpinID (6 bits): identificación del número de pin. En función de *RaspModel* podrá ser un número u otro. En la imagen de la izquierda están los pines de los modelos A y B, a la derecha los pines del resto de modelos. Se identifica por pin en vez del número *GPIO* para poder dar más versatilidad y nivel de acceso al hardware de la *Raspberry Pi*.

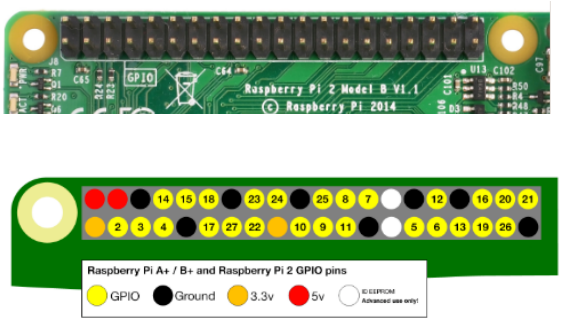

Figura 16. Puertos GPIO Raspberry Pi disponibles en todos los modelos actuales. En los primeros modelos hasta el modelo B+ tienen 26 pines menos [8]

GPIOType (2 bits): corresponderá a alguno de los siguientes componentes:

- 00: *GPIO*, pines de uso general
- 01: *I2C*, protocolo *I2C* y comunicarse con este tipo de hardware
- 10: *UART*, pines para usar este protocolo serie de comunicaciones
- 11: *N/A*

Mode (1 bit): indica lectura (0) o escritura (1) hacia el puerto seleccionado

UARTbit (opcional, 1 bit): bit de comienzo de envío datos serie (1) y final del proceso (0)

ACK (1 bit): comprobación de recepción

GPIOData (opcional, tamaño variable): aquí se almacenará la información tanto de lectura como de escritura.

Conexión:

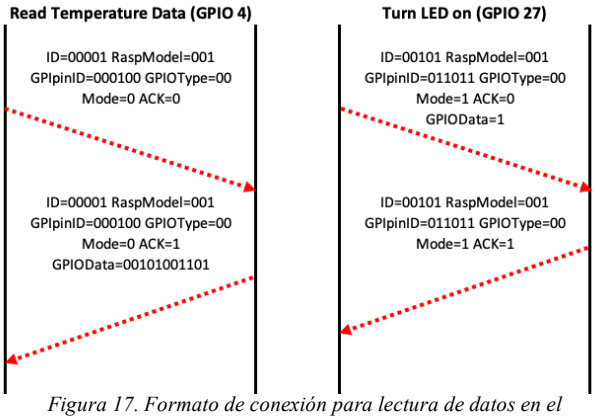

Figura 17. Formato de conexión para lectura de datos en el GPIO 4 (por ejemplo, temperatura) y encendido de un LED por el GPIO 27

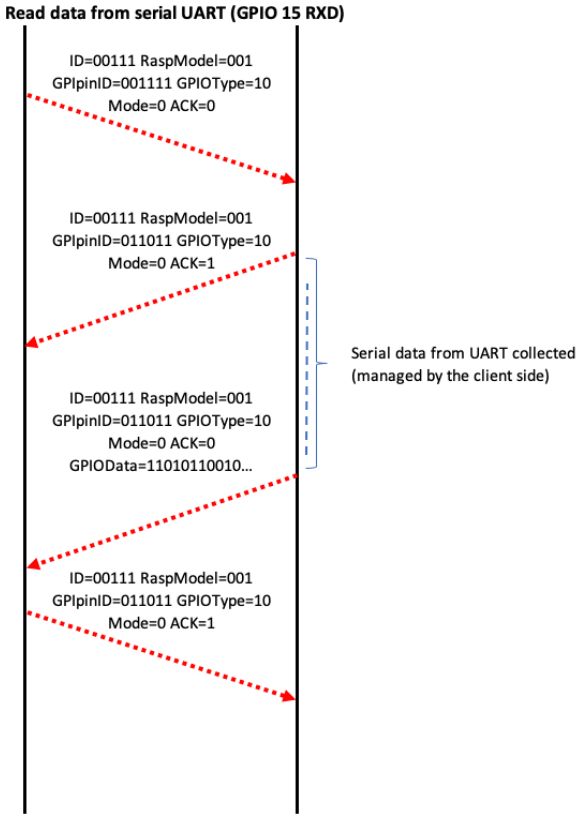

Figura 18. Formato de conexión para lectura de datos serie procedentes de los GPIO UART

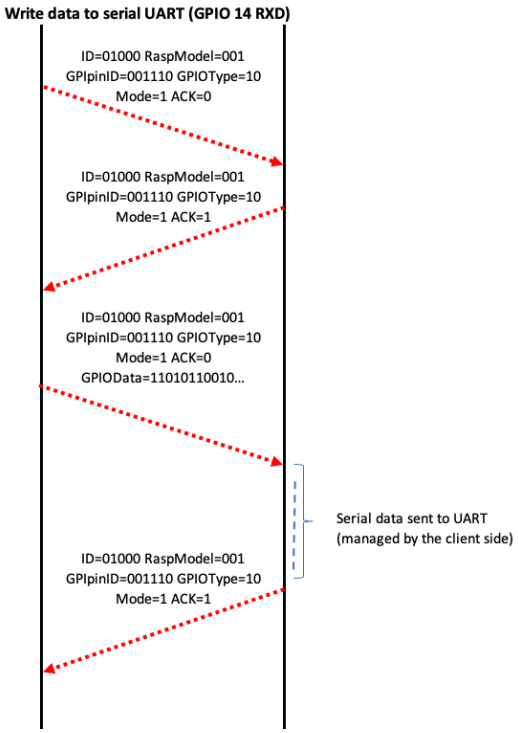

Figura 10. Formato de conexión para escritura de datos serie procedentes de los GPIO UART

4. CONCLUSIONES

La utilización y aplicación práctica del protocolo *SMS* aún tiene mucho que ofrecer, tal y como hemos intentado demostrar con *Stack SMS* en este documento. Este protocolo de mensajes de texto nos ofrece una plataforma perfecta para gestionar cualquier tipo de información en una red de conexión que no dependa de Internet. Además, *Stack SMS* permite interconectar diferentes tecnologías, las cuales en principio parecen no ser compatibles entre ellas. No es una solución optima para enviar gran cantidad de información, sobre todo por la fragmentación en diferentes *SMS*, pero por otro lado sí que permite crear canales de comunicación permanentes (con sólo tener cobertura *GSM*) o temporales para, por ejemplo, enviar órdenes directas, ejecutar pequeñas tareas, etc. Debido a esta forma de funcionamiento, *Stack SMS* se convierte por naturaleza en una buena vía de comunicación para dispositivos *IoT*. Por ejemplo, tal y como se ha podido ver en las aplicaciones descritas en este documento, es perfectamente posible enviar y recibir todo tipo de información y ejecutar una gran variedad de operaciones, como, por ejemplo, temperaturas, registros de todo tipo, activación de relés, motores, etc.

Con simplemente una tarjeta *SIM* conectada a un dispositivo, es posible implementar todo tipo de tareas para ser controladas a través de *SMS* y por lo tanto con Stack SMS. Gracias también al *SDK* de *Stack SMS*, se facilita el trabajo de implementar prácticamente cualquier idea (que pueda funcionar bajo el envío de *SMS*) diseñando simplemente la trama con sus diferentes variantes y luego diseñar una aplicación cliente utilizando el *SDK* disponible, independientemente del lenguaje de programación utilizado. *Stack SMS* ofrece finalmente una solución que permite asegurar que la información enviada es fiable (al igual que ocurre con el protocolo *TCP* del cual hemos intentado aplicar sus principios básicos de resilencia) y que además ofrece seguridad en la transmisión (debido al cifrado de los datos).

5. RECONOCIMIENTO

Trabajo desarrollado por el departamento de Ideas Locas CDO de Telefónica

6. REFERENCIAS

[1] SMS.Technical Specifications. Enlace: https://www.etsi.org/deliver/etsi_ts/123000_123099/123040/12.02.00_60/ts_123040v120200p.pdf

[2] GSM Technical Specification. ETSI.. Enlace: https://www.etsi.org/deliver/etsi_gts/05/0505/05.00.00_60/gsmts_0505v050000p.pdf

[3] W. Stevens, Richard (2001). TCP/IP Illustrated, Vol 1,2 y 3 (20th edición).Addison-Wesley.

[4] Advanced Encryption Standard (AES). FIPSP. Enlace: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

[5] Latch, ElevenPaths., Web oficial: https://latch.elevenpaths.com/

[6] Wikipedia. Juego del Ahorcado. Enlace: https://es.wikipedia.org/wiki/Ahorcado_(juego)

[7] Raspberri Pi. Web oficial. Enlace: https://www.raspberrypi.org/

[8] Wikipedia. Raspberri Pi GPIO: https://www.raspberrypi.org/documentation/usage/gpio/