Capítulo 2

Análisis de tecnologías

2.1. Tecnologías existentes en el mercado

Dado el acelerado crecimiento comercial en la prestación de servicicios móviles basados en texto, existe en el mercado una oferta considerable de soluciones con distintos enfoques y características, mismas que a continuación, serán exploradas de manera general.

2.1.1. Productos y tecnologías

Hay una buena gama de opciones y diferencias entre los distintos productos SMS. La razón por la que se ha escogido Kannel para este proyecto, es que de todos, es uno de los más aceptados a nivel general por las empresas, ya que implementa una gran cantidad de protocolos *smsc* de manera correcta, por ser suficientemente estable en condiciones de producción, y por ser Open Source, lo que implica que es un proyecto constantemente actualizado y con un buen control de calidad, ya que la cantidad de desarrolladores y usuarios es alta, y todos están en constante comunicación.(Tabla 2.1)

Plataforma

El primer punto a explorar es la integración con Java, es decir que un producto esté preparado naturalmente para interactuar con Java o no; en este caso, Kannel no lo está. De ahi nace la necesidad de crear un gateway entre el protocolo nativo de Kannel y Java, para dicho efecto, es necesaria la implementación de un API que pueda descifrar y codificar dicho protocolo.

J2SE es la versión estándard de java, que cuenta únicamente con las herramientas para que un programa use la máquina virtual (JVM).

Comunicación entre aplicaciones/componentes

Éste es un punto clave en el diseño de un producto SMS, ya que de ello depende en gran medida la flexibilidad que pueda llegar a tener una aplicación soportada por éste.

Actualmente, el modelo más popular por su facilidad de implementación es HTTP, que padece de los problemas descritos en el capítulo 1.

Un método que aún no ha sido explotado por completo, y que resuelve muchos de los problemas presentados por HTTP son las colas de mensajes (MQ). Cabe señalar, que para este trabajo, la atención será en particular para el servicio de mensajes de Java (JMS).

Open Source

El tipo de licencia de un producto es importante por que de éste depende el alcance del proyecto y el costo. En nuestro caso, Kannel usa una licencia tipo BSD que no restringe ni el tipo de uso del software ni del código fuente del mismo, siempre y cuando se conserve dicha licencia, a diferencia de otros productos que son caros y de accesso restringido al código fuente para desarrollo experimental.

	J2SE	J2EE	JMS	HTTP	WebService	OpenSource	Dist.	Sist.Oper.
Kannel				X		Tipo BSD	X	Multi
Provato NCL	X	X	X	X	X	No	X	Multi
Now SMS				X		No		Windows
Donald				X		No	X	Windows
EMS	X	X	X	X	X	No	X	Multi

Tabla 2.1: Relación entre algunos productos similares a Kannel, tecnologías y otras características relevantes

2.2. Protocolo de intercambio de información y control en Kannel

Para lograr comunicar cada módulo, Kannel implementa un sencillo protocolo de propósito especifico basado en estructuras (struct) de C(Ver A.1). Para realizar un análisis de este protocolo, primero fue necesario descifrarlo, para generar una especificación del mismo (Apéndice A).

2.2.1. Método de desciframiento

El problema para usar el mismo protocolo que Kannel usa, radica en que no existe documentacion del mismo, nadie más que él que lo desarrolló sabe exactamente cómo es que éste de hecho funciona. Entonces, el siguiente paso fue preguntar en la lista de desarrolladores, pero al hacerlo nadie pudo responder con un docuemnto detallado al respecto y sugirieron investigarlo en el código fuente de Kannel.

Ingeniería inversa con código fuente

Para descifrar el protocolo usando el código fuente, hubo que ir rastreando y corriendo el código a mano hasta hallar los archivos y líneas que manejan en primer lugar la representación de los datos, lo cual fue encontrado en los archivos `msg.c`, `msg.h` y

`msg-decl.h`. Una vez hallados estos archivos hubo que ejecutar el preprocesador de C para llegar a una definición de estructuras de C entendible por el programador promedio (Ver A.1).

Ingeniería inversa con sniffer

Para observar en realidad como es que se comportan los paquetes del protocolo en el medio usando un sniffer (Figura 2.1), lo primero que debe hacerse es identificar un paquete correspondiente al protocolo, en este caso se observa el primer paquete enviado por un smsbox al iniciar una sesión con el bearerbox :

```
0x0000   4500 0044 791a 4000 4006 c397 7f00 0001        E..Dy.@.@.......
0x0010   7f00 0001 80a4 1a0b 1c28 15c3 1c80 99e0        .........(......
0x0020   8018 7fff 9bb5 0000 0101 080a 0017 6cda        ..............l.
0x0030   0017 6cda 0000 000c 0000 0001 0000 0003        ..l.............
0x0040   ffff ffff                                      ....
```

Los primeros 20 bytes corresponden al encabezado IP:

```
4500 0044 791a 4000 4006 c397 7f00 0001
7f00 0001
```

Los siguientes 32 bytes, corresponden al encabezado TCP:

```
80a4 1a0b 1c28 15c3 1c80 99e0 8018 7fff
9bb5 0000 0101 080a 0017 6cda 0017 6cda
```

Y finalmente los ultimos 16 bytes, corresponden al paquete de identificación de Kannel :

```
0000 000c 0000 0001 0000 0003 ffff ffff
```

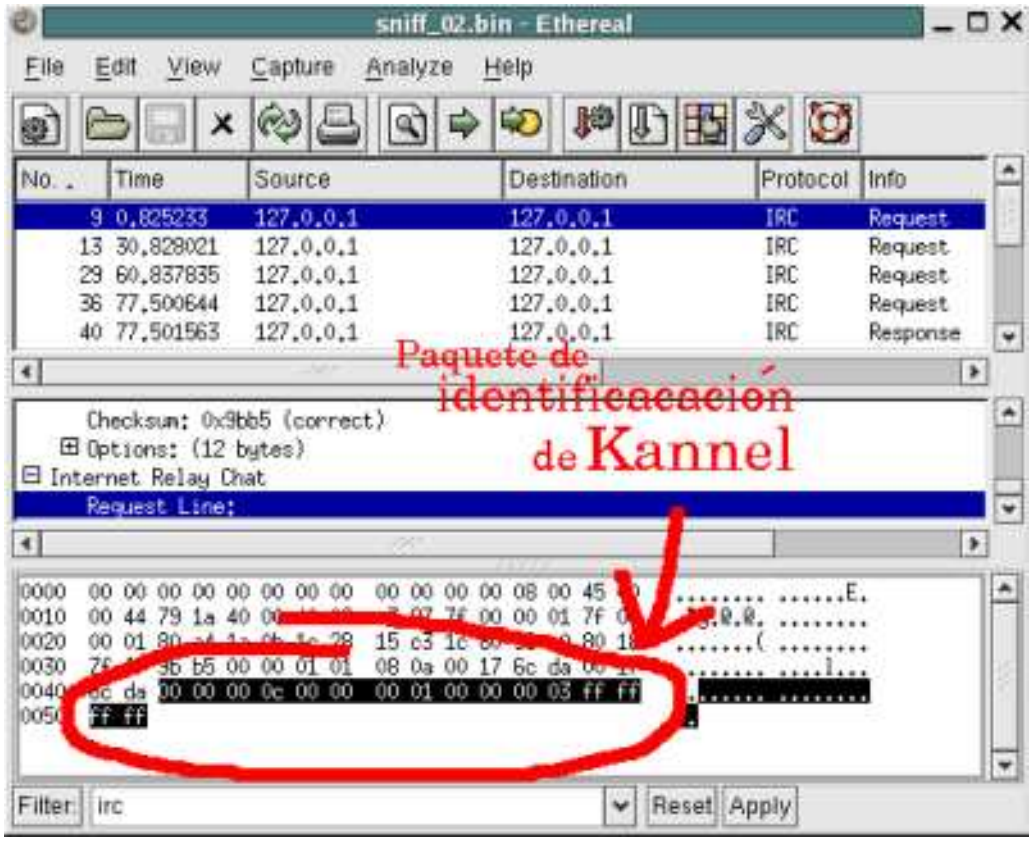

Figura 2.1: Paquete de identificación de un smsbox a un bearerbox desplegados en una ventana de ethereal

Donde, los primeros 4 bytes, corresponden al tamaño del paquete: 0000 000c = 13

Los siguientes 4 bytes, corresponden al tipo de paquete: 0000 0001 = 1 = admin

Los siguientes 4 bytes, corresponden al comando solicitado: 0000 0003 = 3 = Identification

Y los 4 bytes restantes al identificador interno del smsbox.

2.2.2. Descripción General del protocolo

Este protocolo mantiene conectados dos módulos de kannel entre sí, permitiendo la transferencia de datos y paquetes entre ellos.

Una sesión es iniciada cuando un módulo cliente solicita conectarse al bearerbox enviando un comando administrativo que solicita ser identificado. La misma es terminada cuando el bearerbox solicita al cliente ser apagado o reiniciado.

Durante la sesión, ambas partes pueden enviar latidos (heartbeats) entre sí para indicar a la otra parte que el enlace está activo y la aplicación esta viva.

En una sesión con uns smsbox cuando un smsbox ha establecido una sesión, puede enviar y recibir packetes tipo SMS.

En una sesión con un wapbox cuando un smsbox ha establecido una sesión, puede enviar y recibir datagramas WDP (Wireless Datagram Protocol).

Nota: Para mayores detalles sobre el protocolo ver el apéndice A

2.2.3. Análisis del protocolo

La intención original, fue hacer una *especificación* del protocolo que usa Kannel, pero después de descifrarlo y estudiarlo, he creído que no es apropiado nombrarle *especificación*, sino nombrarle *descripción* (Apéndice A), ya que la única instancia del protocolo se encuentra implícita dentro del código fuente, sin una estructura clara del mismo que pudiera fácilmente ser descrita usando pseudocódigo.

Seguridad

El protcolo no soporta actualmente mecanismos formales de autenticación más que por medio de la dirección IP, pero este tipo de autenticación no forma parte del protocolo, sino de la implementación. Otra forma en la que puede autenticarse una conexión por medio de este protocolo es mediante el uso de tecnologías como SSL (Secure Socket Layer's), pero éste sigue estando fuera de los verdaderos alcances del protocolo y es también un factor de implementación.

Desempeño

El planteamiento estructural del protocolo es apropiado, pero está implementado usando tipos de datos nativos de C, lo cual hace que la cantidad de información transmitida sea tan excesiva, que algunos paquetes llegan a ser casi 3 veces más grandes de lo que realmente necesitan ser. Tomemos por ejemplo, el paquete de la sección 2.2.1:

```
0000 000c 0000 0001 0000 0003 ffff ffff
```

Si lo agrupamos por campos,

```
0000000c 00000001 00000003 ffffffff
^------^ ^------^ ^------^ ^------^
Longitud  Tipo    Comando     ID
```

podemos ver que dado el tipo de aplicación:

La longitud, nunca llegará a ser mayor a unos cuantos cientos de bytes, para los casos más extremos, siendo el promedio de unos cuantos cientos de bytes, por lo que el uso de un entero con signo de 4 bytes es excesivo, el uso de un entero con signo de 2 bytes, estaría muy holgado en este campo.

Los tipos de paquetes actualmente definidos son 5, por lo que 3 bits, serían suficientes, el problema en dicho caso, sería la necesidad del uso de bits de relleno para transmitirlo con facilidad; por esto, sería conveniente la asignación de 4 bits (medio byte) para dicho campo, lo cual nos daría una buena holgura para siguientes diseños basados en este protocolo.

Comando, la situación es la misma que la anterior. Si para este caso asignáramos otra vez 4 bits, podríamos fusionar el campo anterior con este para tener un byte completo para ambos campos.

ID, se refiere al identificador del cliente que se conecta al bearerbox. Para éste tenemos que el número real de clientes que tendría el bearerbox raramente sería mayor a una docena, pero si suponemos que todos los servicios SMS del mundo fueran a concentrarse a través de un bearerbox, y que cada país necesitara 10 clientes bearerbox para satisfacer la demanda (ambos casos excesivos), tomando en cuenta el conteo actual *(*2004*)* de países miembors de la ONU (para tener un número más o menos estable), tendríamos la necesidad de asignar $191 * 10 = 1910$ identificadores lo cual puede hacerse en con 11 bits; si esta cantidad la redondeamos a 16 bits, para tener un identificador de 2 bytes, tendríamos espacio para asignar 65535 identidicadores, mucho más que lo que hace un momento creíamos suficiente.

Tomando en cuenta estos cálculos, podríamos mejorar esto:

```
0000 000c 0000 0001 0000 0003 ffff ffff
```

en esto :

```
000c 13 ffff
```

una formación optimizada del paquete, de 5 bytes contra 16 bytes.

Después de analizar este protocolo, es fácil notar que es mejorable en muchos aspectos, pero a pesar de todo puede ser usado para nuestros propósitos, con mínimas o ninguna adaptación tanto del bearerbox como del protocolo.