COFOCUS
Compact and Expanded Restful Services for Mobile Environments

Li Li and Wu Chou
Avaya Labs Research, 233 Mount Airy Road, Basking Ridge, New Jersey 07920, U.S.A.

Keywords: REST web service, Mobile device, Compact HTTP, XMPP, SMS, Android, Security.

Abstract: In this paper, we present an approach to enable collaborative endpoint network for mobile phones. In particular, we expose functions on mobile phones as REST web services that make mobile phones as web service providers for rapid integration with communication and collaboration applications. Because mobile phones have distinct features and constraints, this paper describes a lightweight and efficient protocol, Compact HTTP, which consists of a small subset of HTTP 1.1 to reduce the footprint of REST services. We expand bindings of HTTP to multiple messaging protocols, including SMS and XMPP, and make the REST services invariant to network and protocol changes. These expanded bindings enforce asynchrony into REST, a desired property for many communication and collaboration services. Furthermore, HTTP over XMPP described in our approach introduces the concept of hyperlink presence in collaboration, and it is used to mitigate the broken link issue which is critical in mobile environments. To provide end-to-end message security, a symmetric key based security scheme is described for service authentication and authorization. A prototype system based on the proposed approach is developed that allows both local operators and remote directors to control and monitor the camera, camcorder, location, telephony, motion, power, etc. on Android phones in a secure manner. Experimental results indicate that the proposed approach is feasible, lightweight, and has satisfactory performance.

1 INTRODUCTION

As mobile phones become more and more advanced, they are replacing other devices, such as PDA and notebook computers, as the next generation personal digital assistant. Compared to other computing devices, the mobile phones offer a unique combination of telephony functions (making and receiving phone calls), sensory functions (sound, camera, camcorder, location, acceleration, temperature, etc.), and communication networks (3G, WiFi, Bluetooth, Wimax, etc.). More and more applications are designed for mobile phones to take advantage of these capabilities, as evidenced by the popularity of iPhone and Android applications in their App Stores.

The focus of this paper is on a web service approach to enable collaborative endpoint network for mobile phones and to expose functions on mobile phones as REST web services, such that applications running remotely can monitor and control them in near real-time manner. Collaborative endpoint network is an emerging area with critical applications in intelligent home network, etc. There are many motivating use cases to extend the collaborative endpoint network to mobile phones. For example, we can use a mobile phone as a surveillance device to monitor a room or a car. Mobile phones can also be used in healthcare to monitor and remind patients of their treatments, as well as find and locate medical professionals to treat them. A travel application is to use mobile phones as the virtual tour guide and push relevant multimedia content to a visitor as he moves around a tourism site. Mobile phones are also an ideal device to keep track of traffic flows when the speed of many drivers can be obtained and aggregated automatically. All these applications require the ability to monitor and control one or more functions on the phone in near real-time, as these functions, such as location, may change frequently and in-time responses are needed.

An efficient approach to support these applications is to expose the fundamental functions on the phones as REST web services and make mobile phones as web service endpoints, so that services on mobile phones can be invoked and composed in different ways by different

Li L. and Chou W..
COFOCUS - Compact and Expanded Restful Services for Mobile Environments.
DOI: 10.5220/0003346200510060
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 51-60
ISBN: 978-989-8425-51-5

applications. This approach eliminates the need for each application to duplicate the same function. Making a phone into a web service endpoint enables the applications to interact with the phones in heterogeneous mobile environments, as web service is independent of transport protocols and programming languages. REST web service is easy to extend as it supports dynamic discovery through links. For example, to add a second camera on the phone into the services, we just need to implement a new camera resource and link it to the main resource.

There are two types of web services as we know it: SOAP based (SOAP 2007) and REST based (Fielding 2000). Many approaches have chosen REST based web services for mobile devices because of its simplicity and close relationship with the architecture of Web. We chooses REST in our approach for the same reason. However, we found that HTTP 1.1 protocol as used in current REST web services needs both compaction and expansion in mobile phone environments. First, we need to compact HTTP 1.1 messages as they can be complex and large while some features are never used in mobile phones. Second, we need to expand HTTP to multiple transport protocol bindings besides TCP/IP to support REST services in heterogenous mobile environments. To address these issues, we propose and define a "Compact HTTP" protocol, consisting of a small set of HTTP 1.1 while keeping only the essential elements of HTTP 1.1 to enable collaborative endpoint network of mobile phones.

Moreover, we describe how Compact HTTP can be bound to multiple messaging protocols, in particular to XMPP (XMPP 2004) and SMS (Short Message Service). These protocol bindings introduces asynchrony into REST to support event-driven REST web services on mobile phones. Furthermore, HTTP over XMPP in our approach introduces hyperlink presence into REST to mitigate the broken link issue which is critical for mobile phones. A security protocol is also devised to permit flexible and quick setup of security contexts between services and clients. Based on this protocol, we develop a lightweight REST web services framework on an Android phone. Within this framework, we implement a few dozen resources, including sound, camera, camcorder, location, power, motion, scheduler, and telephony manager as secured REST web services. Our collaborative endpoint network framework also supports web storages, including Google Sites, YouTube, etc., to upload recorded media for instant sharing and collaboration.

The rest of this paper is organized as follows. Section 2 surveys some related work in developing REST web services for mobile devices. Section 3 describes the Compact HTTP protocol. Section 4 describes the binding of this protocol to multiple transport protocols. Section 5 presents a security protocol to address related security issues. Section 6 describes the implementation and experiments of a prototype systems running on a live wirless carrier network (T-Mobile). And we conclude and sumarize this paper in Section 7.

2 RELATED WORK

Because of its relative simplicity, compared to SOAP based web services, REST web service paradigm is gaining popularity in mobile device communities.

The principles and architectures of REST web services are extensively discussed in (Fielding 2000) and (Richardson 2007), which are followed by this paper whenever applicable.

WAP (WAP 2001) is a suite of protocols to connect wireless mobile phones to the Web. The typical WAP architecture consists of Content Server, WAP Gateway and mobile devices. When requested by the Content Server, the WAP Gateway uses a protocol WSP (WSP 2001), which is a binary version of HTTP, to transfer encoded WML content (WML 2001) to the devices. However, modern smart phones rarely support WAP as they can interpret HTML directly.

Constrained REST Environments (Core) (Core 2010) is a recent IETF activity to restrict HTTP to host resources on low-end devices, such as 8-bit microcontrollers with up to 12 KB of memory. It proposes a binding of HTTP to UDP that deals with asynchronous messages. However, this approach is not suitable for mobile environments, as the phones do not have a reachable IP address, a major issue that has to be addressed properly.

MacFaddin et al. (MacFaddin 2008) proposed a REST web service framework for mobile commerce spaces.

Liu and Conelly (Liu 2008) proposed to combine REST web service with semantic web technology to support services on mobile computing units.

Lozano et al (Lozano 2008) promoted the use of REST web services to expose IMS capabilities to Web 2.0 applications. In particular, it proposes the use of AtomPub protocol to publish the IMS resources.

Antila et al (Antila 2009) discussed the hosting of REST web services on mobile devices to support person-to-person communication and collaborations over Wifi and 3G networks.

Aijaz et al (Aijaz 2009) presented a REST web service provisioning system and compared it against the SOAP counterpart. The experiments showed that REST messages have much lower overhead than SOAP messages.

Pruter et al. (Pruter 2009) described an approach to adapt resource-oriented service framework to automatically control robots for industrial automation tasks. It uses a special mechanism called MIRROR to handle events. Performances of their framework are evaluated under three physical networks: wireless LAN, Bluetooth and ZigBee. The results showed that the REST framework has lower overhead than SOAP based DPWS.

AlShahwan et al (AlShahwan 2010) compared the performances of the same SOAP and REST services implemented on mobile devices. They conclude that REST is more suitable to mobile devices as it requires less processing time and memory.

Stirbu (Stirbu 2010) presented a REST architecture to render web interfaces on mobile devices in which REST protocol is used to synchronize states between applications.

However, none of the abovementioned prior work has studied how to compact HTTP for mobile devices and how to expand HTTP to connect phones with each other to enable a collaborative endpoint network in mobile phone environment and to link functions on the phones with enterprise applications.

3 COMPACT HTTP

HTTP 1.1 has a set of very powerful features that can be too expensive to implement and support on resource-constrained mobile devices. Even though it is possible to run a HTTP 1.1 server on Android mobile phones, many of its features may never be utilized. Therefore, we elect to develop a lightweight compact protocol, compact HTTP, for mobile phones. It consists of a subset of HTTP 1.1 protocol, but is still capable to enable the endpoint network and collaborative applications of mobile phones. Protocol compaction in our approach is only a process not the final goal – as mobile phones become more powerful, less compaction will be needed. For this reason, we choose to represent HTTP message in plain text, instead of binary, as this compact subset of HTTP 1.1 may grow to support further extensions.

3.1 Message Templates

To reach a compacted subset, we start from an empty feature set and add features to it as necessary until the desired services are covered. In our case, this exercise leads to the following Compact HTTP request and response templates that follow HTTP 1.1 closely:

```
{operation} {path} {version}
Authorization:{token}
x-mid:{string}

{form}
```

```
{status} {reason} {version}
Authorization:{token}
x-mid:{string}

{form}
```

Figure 1: Message Templates for Compact HTTP request (above) and response (below).

All the variables, including `{operation}`, `{path}`, `{version}`, `{status}`, `{reason}`, are as defined in HTTP 1.1. For Compact HTTP, the version is `HTTP/1.1c`. `{form}` is defined by HTML 4 (HTML 1999), whose media type is `application/x-www-form-urlencoded`. The differences with HTTP 1.1 are described below.

`Authorization` contains the access token for the message. In HTTP 1.1, this is a request header. We extend it to responses because they may be sent in a different connection. Therefore, the client needs to authorize a response before taking any action (for example, update user interface).

`x-mid` is a new header to HTTP 1.1 for clients to correlate asynchronous responses and events to requests. Its value is set in a request and echoed in the responses.

The templates omit some headers considered important for REST services, such as the `ETag` response header. The reason is that our resources tend to have small representations (measured by the size of its form) that can be updated and transmitted without checking the versions. We also omit content negotiation headers in favour of using URI. Using URI to identify media type is an approach recommended by Richardson (Richardson 2007).

3.2 Message Exchange Patterns

Typical HTTP messages follow the request-response pattern. Services following this pattern are *atomic* as the service is complete once the response is sent. However, many services on the mobile phones are multi-step instead of atomic. For example, to control the camera to take a picture involves the following steps: 1) adjust focus; 2) take shot; 3) upload picture to a web site. Modeling this type of services as atomic would not be a scalable approach because: 1) it makes the service stateful, which violates the REST statelessness principle; and 2) it makes a client less responsive to service state changes and it is much more expensive to recover from faults.

A solution in our approach is to model the multi-step services as one request followed by multiple responses, in which intermediate responses indicate service progression while the final one indicates service completion. To separate them, the status for the intermediate responses should be `202 Accepted` or `206 Partial Content`, while the final response should be `200 OK` or `201 Created`. Status `206` is used only with `GET` in HTTP 1.1, and here we extend it to any request to convey the current service state as partial content. All asynchronous responses are correlated to its request by `x-mid` for both atomic and multi-step services.

Another important type of message exchange pattern is event subscription and notifications. For example, a client can subscribe to a phone's location tracking service to receive notifications about location changes of the phone. In this pattern, a subscription (sent by a client as `PUT`) is followed by unknown number of notifications (sent by the service as `POST`). It is needed for a client to tell which notifications are from which subscriptions, so that it can adjust the subscriptions, e.g. cancel, etc. To support this feature, the `x-mid` of notifications would echo the `x-mid` of the subscription request.

3.3 Design Patterns

The Compact HTTP does not specify how to design the resources to support these message exchange patterns. For these, we suggest to follow the REST service design patterns (Li 2010), including session, event subscription, multi-resource and multi-state, that are common in real-time communication services.

4 COMPACT HTTP BINDINGS

In conventional REST web services, there is a basic and implicit assumption that a HTTP server has a public IP address. However, this is not possible in mobile environment, as a mobile phone is typically behind its provider's NAT gateway and its private IP address is not reachable from outside. The enterprise applications that control and monitor phones are also behind corporate firewalls. This creates an issue for many real-time applications where two-way messaging is required. On the other hand, many 3G mobile phones can join different communication networks over different protocols such that they are reachable without IP addresses. Instead of IP address, a phone can be addressed by a phone number (SMS), an email address (SMTP) or JID (XMPP).

Therefore, to support REST services in these heterogeneous environments, it is necessary to decouple HTTP from TCP/IP. In addition to TCP/IP, it is both convenient and advantages to treat these messaging protocols as "transport" for HTTP. This approach makes the REST services invariant to the protocol changes as the mobile phone connects to different networks. This allows us to keep the same services while optimizing their performance over available networks and protocols. For example, we may choose to transmit time sensitive messages over TCP/IP or XMPP, and noncritical ones over SMS or SMTP. Furthermore, HTTP over XMPP can bring presence information into REST architecture. In particular, we can assign presence to hyperlinks to address the issue of broken links in the Web.

The idea of separating protocol messages and transport is not new. It is actually one of the tenets of web service paradigm that web services should be agnostic to transport protocols. SOAP based web services community has embraced this approach by defining SOAP bindings to HTTP (WSDL 2001), to XMPP (SOAP/XMPP 2005), and to JMS (SOAP/JMS 2009). However, HTTP has bindings only to TCP/IP and UDP, as far as we know.

The following subsections describe the components as well as process of HTTP bindings with other message transport protocols. In particular, we focus on the binding of HTTP with XMPP in our approach.

4.1 URI Scheme

In REST services, any resource must have a URI that identifies where the resource is and how to contact it. Because we want to bind HTTP to different transport protocols, we adopt a two-level URI scheme according to (RFC 3986), where the first level URI identifies the HTTP information and the second level URI identifies the transport information:

```
    uri_1 = http://{uri_2}/...
    {uri_2} = URI
```

For example, to address a resource `x` with HTTP over SMS to a phone number, we can use:

```
            http://sms:5555/x
```

Or alternatively, through a SMTP gateway as:

```
   http://smtp:5555@example.com/x
```

To address the same resource with HTTP over XMPP, we would use:

```
    http://xmpp:joe@example.com/x
```

If the phone has an IP address, the URI for the resource would simply be:

```
            http://123.4.5.6/x
```

To communicate with a resource with `uri_1`, a client first establishes a communication channel according to `uri_2` and then transmits HTTP messages over the channel. This process is elaborated in the next section using XMPP as an example.

4.2 XMPP

XMPP architecture consists of XMPP servers and clients. To exchange messages, clients have to establish a TCP/IP connection to the same or federated XMPP servers. There are many open XMPP servers, including Google Talk, that give people free accounts with federated identities. For instance, you can use a valid GMail account to login to the Google Talk server.

To connect to a XMPP server, A XMPP client typically needs to know the following information: 1) XMPP host and port (talk.google.com:5222); and 2) XMPP Service (gmail.com). This information is not included in URI scheme because a XMPP client always connects to one XMPP server in a collaboration session. Therefore, the information can be saved in a configuration file associated with a user. Once the user logs into the XMPP server, the established connection is used to transmit all HTTP messages.

Because chat is in the base protocols of XMPP and most XMPP clients support this feature, we choose to transmit Compact HTTP messages as chat over XMPP. The template for HTTP request and response over XMPP is:

```
<message
   from='{from_jid}' to='{to_jid}'>
   <body>
     {Compact HTTP message}
   </body>
</message>
```

4.3 Hyperlink Presence

In the Web, there is an annoying issue of broken hyperlinks as the resource that embeds the links is not aware of the linked resource. When the hosted linked resources on the web server shuts down or the resource is moved or deleted, the link becomes broken. The problem is that it is difficult for the client to know when the link will be restored unless it polls the server constantly, which creates unnecessary network traffic.

Our approach of HTTP over XMPP offers a solution to this problem by using the presence services provided by the XMPP layer. If a XMPP client listens for presence updates from a JID address, it can assign presence to a hyperlink containing that address in near real-time. For example, to assign presence to a hyperlink: `http://xmpp:someone@gmail.com/x`, the client just needs to monitor the presence of `someone@gmail.com`. By knowing the link presence, the client can avoid fetching or polling the broken links all together. The presence also gives the REST services the ability to change their presence, for example in case of temporary maintenance, while keeping the clients informed in a timely fashion.

4.4 SMS

SMS is a very common feature in most mobile phones. There are several ways to send and receive SMS messages depending on the networks. In the cellular networks that provide SMS, two phones can exchange SMS directly. Outside the cellular networks, a client could use a SMS gateway that exposes particular protocols, such as SMTP to send and IMAP to retrieve SMS messages. This approach is available from T-Mobile who assigns each G1 phone an email address that ties with its number: `{number}@tmobile.net`.

To bind to SMS, the Compact HTTP messages are contained in the UD (User Data) segment of SMS PDU (SMS 2010). To bind to SMTP, the HTTP message is contained in the Email body.

5 SECURITY

Security is a critical issue in hosting web services on mobile phones because the services open access to resources on the phone that may be misused by malicious clients. It is typical to deploy layered security mechanisms from protocol up to applications such that a breach at lower level can be defended by the layer above. In our case, security mechanisms at Compact HTTP, transport protocols, services and applications are employed. The following sections will discuss these mechanisms in more detail.

5.1 Secure Messages

Because in mobile environments, a HTTP message may travel through different networks in different protocols, it is necessary to employ end-to-end message level security. Here we outline a symmetric key based security protocol to set up the security contexts between two parties.

The design goal is to allow the user who operates the phone to quickly grant and reject clients and service requests. For this reason, we chose symmetric key because our applications are aimed at a group of trusted users that can exchange secret keys easily. In some cases, the phone and the client are managed by the same user.

For convenience, the user that configures the phone is referred to as "operator" and the user that configures the application is referred as "director." The protocol consists of the following steps:

1. The operator and director agree upon a secret passphrase `P` and enter it into the phone and the client respectively.

2. The director creates an access token `T1` and tells it to the operator.

3. The operator enters client's URI `A` and `T1` into the phone and creates an access token `T2`. The token is sent by a "join" message encrypted by `P` to URI `A` as follows:

```
PUT operators/{phone} HTTP/1.1c
Authorization: T1
x-mid: {number}

token=T2
```

4. The client decrypts the message with `P`, store `T2` and sends an encrypted response message that indicates acceptance.

5. The phone receives and decrypts the response and activates the security context, which contains `(A, P, T1, T2)`.

6. Any subsequent message from the client to the phone will contain `T2` and be encrypted with `P`. The phone will decrypt any message from `A` with `P` and checks it against `T2`. Any response message to `A` will contain `T1` and be encrypted with `P`.

7. The operator deactivates the security context by sending a "Delete" message to the client. The corresponding security context becomes invalid on the phone and client:

```
DELETE operators/{phone} HTTP/1.1c
Authorization: T1
x-mid: {number}
```

Figure 2 illustrates the phone interface for the operator to carry out steps 3-5 and step 7.

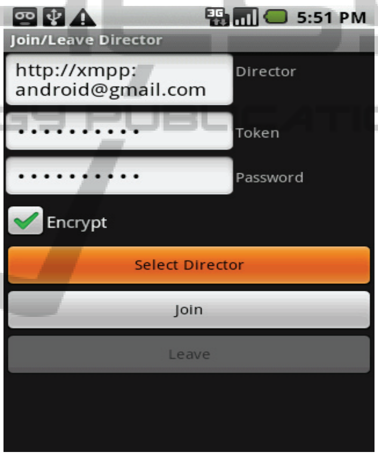

Figure 2: Screenshot of Join/Leave Director Activity.

5.2 Secure Services and Data

Because a mobile phone that hosts REST services is also used for other purposes, the operator can start and stop services, login and out of transport services, as ways to control access to the services.

In our system, we further limit message exchange patterns for security reasons:

- The respond messages are always sent back to the requester on the same transport protocol, who has been authenticated and authorized.
- There is only one subscription for each resource and the event listener must be the same as the subscriber, who has been authenticated and authorized.

Our system also secures the access to sensitive data collected by the services, such as captured images and videos. Instead of returning the content to the authenticated client directly, the system stores it locally or stores it in a web storage site and provides a link to the client, who can retrieve it with another set of credentials.

6 PROTOTYPE SYSTEM

Based on the proposed protocols, we developed a prototype system that hosts a set of REST resources on T-Mobile G1 phones running Android 1.6 in T-Mobile cellular network. The intended relations between various client applications and the phones are illustrated in Figure 3 where REST services are accessible to clients of different kinds over the heterogeneous networks.

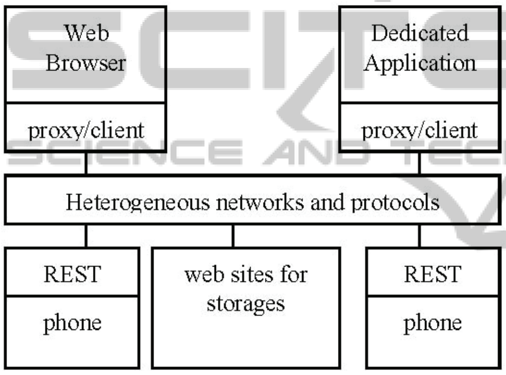

Figure 3: High level relations between client applications and REST services on the phones.

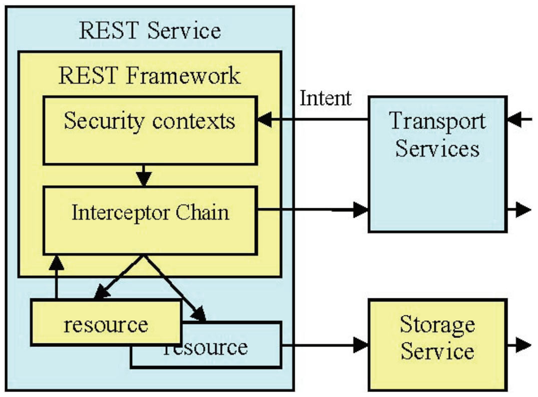

Figure 4: High level architecture of REST service framework on Android phone.

The high-level REST server architecture on an Android phone is illustrated in Figure 4, where the components in blue colour depend on the Android SDK whereas the components in yellow colour depend only on Java SDK. The core REST framework, including the security package, does not import any Android packages and can be run in any Java runtime.

There are three types of Android services in the framework: transport services, REST services and storage service.

The transport services are responsible to listen and send messages over a transport protocol, such as SMS or XMPP. For XMPP transport, we used a XMPP client library compiled by Srinivas (Srinivas 2008) for Android. For SMS, we used Android `SmsManager`. If an incoming message is intended for the REST Service, it is forwarded to the REST Service as an Intent on Android platform.

The REST Service is an Android Service that contains the REST framework. Upon start, the REST Service registers an Intent Listener. Upon receiving a message encapsulated in the Intent, the Intent Listener looks up the security context for the message and invokes the interceptor chain in the REST framework to process the message.

The REST framework contains an incoming interceptor chain, a pivot, and an outgoing interceptor chain. The incoming chain consists of three interceptors to 1) decrypt; 2) deserializes; and 3) authorize the message. If any of these interceptors fails, the message is discarded and an error message is returned. If the message is a request, the pivot will invoke the corresponding resource; if the message is a response, the pivot forwards it to the Android Notification Service. The outgoing chain consists of four interceptors to 1) endorse; 2) serialize; 3) encrypt; and 4) deliver the responses. The chains can also be invoked by a resource to control another resource. For example, a scheduler resource controls the camera by going through the same incoming chain for security reasons.

For message encryption and token generation, we used Java Crypto packages (`javax.crypto` and `java.security`) with password based encryption algorithm `PBEWithMD5AndDES`. The encrypted messages are encoded as Base64 strings for transmission.

The REST resources in our approach can use web storages provided by Google to upload captured audio (Google Docs), image (Picasa) and video (YouTube) contents, so that they can be instantly shared for collaboration. The storage service is an abstraction of local and web storages with three methods: `login(account)`, `logout()`, and `save(uri, content)`. It uses a set of HTTP clients to upload multimedia contents to the designated server and publish the services to a web proxy (e.g. Google Sites). The phone interface to manage web storage is illustrated in Figure 5.

6.1 Implemented Resources

This section lists the major resources developed so far within our collaborative endpoint network framework. Each resource is described by one table that defines its path, service, operations and response patterns, where “a” stands for atomic or “m” stands for multi-step (Table 1).

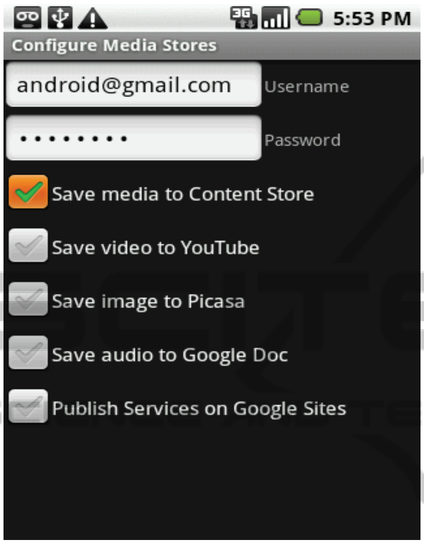
5:53 PM
Configure Media Stores
Username: android@gmail.com
Password: ••••••••
Save media to Content Store ✓
Save video to YouTube
Save image to Picasa
Save audio to Google Doc
Publish Services on Google Sites

Figure 5: Screenshot of Web storage Activity.

Table 1: Implemented resources.

path	**/sound/control**	
GET	Retrieve current state.	a
PUT	Record or playback sound.	m
path	**/camera/control**	
GET	Retrieve current state.	a
PUT	Take or display a picture.	m
path	**/camcorder/control**	
GET	Retrieve current state.	a
PUT	Record or playback video.	m
path	**/location, /power**	
GET	Get current geo location or battery power status.	a
path	/**{source}/monitor**, where {source} is one of gyroscope, light, location, magnet, motion, orientation, pressure, proximity, temperature, phone, and power.	
GET	Get current subscription	a
PUT	Subscribe for events from {source}.	a
DELETE	Unsubscribe events.	a
path	**/schedule**	
POST	Schedule a task in future. For example, start and stop camcorder at given time.	a

6.2 Implemented Clients

Two types of clients were implemented: the Android phone (both client and service) and a dedicated Java desktop client. The Android phone client can control and monitor other Android phones running REST services using HTTP over SMS or XMPP. The Java client, based on open source Smack 3.1.0 XMPP library (Smack API 3.1.0), controls and monitors REST services using HTTP over XMPP only.

6.3 Experimental Results

Our main concern is the performance of our REST based collaborative endpoint network framework. On the server side, we measured the total processing time (from entering the incoming chain to leaving the outgoing chain) as well as the processing time of individual interceptors. On the client side, we measured the round-trip latency (which includes the network latency and XMPP library processing time). To obtain the time, we used the XMPP java client on a desktop computer (Dual Core CPU 3.00 GHz with 2GB RAM) connected to the Internet to send 52 HTTP messages over XMPP to each REST service on a T-Mobile HTC G1 phone (Firmware 1.6, Build DMD64) registered in T-Mobile cellular network. The measurements on the phone were collected using Android `TimeLogger` utility and on our Java desktop client using Java `System.nanoTime` function. The performances are summarized in Table 2 and the message sizes are listed in Table 3.

Table 2: Total client time, total server time and individual interceptor times (ms).

Process	mean	std	min	max
Client	1400.6	821.7	554.3	3858.3
Server	209.53	86.08	135	590
Encrypt	66.05	25.5	47	153
Decrypt	39.12	19.49	2	84
Deserialize	23.98	25.1	2	196
Serialize	7.4	7.3	4	41
Authorize	1.77	0.53	1	3
Endorse	2.4	3.03	1	24
Pivot	40.33	58.98	4	254
Deliver	28.42	3.8	20	47

Table 3: Message Sizes (byte).

Type	mean	std	min	max
Encrypted	113.36	28.6	88	192
Decrypted	77.93	21.54	56	138

The bar graph in Figure 6 illustrates the mean times spent in the interceptors. The graph shows that, on average, encryption, decryption and pivot are the top three time consuming components. They take 69% of total server time. Notice that the pivot time is the mean time spent by the resources to execute HTTP methods, which is outside the control of the framework. Table 3 shows that the small messages are sufficient to support a variety of basic services.

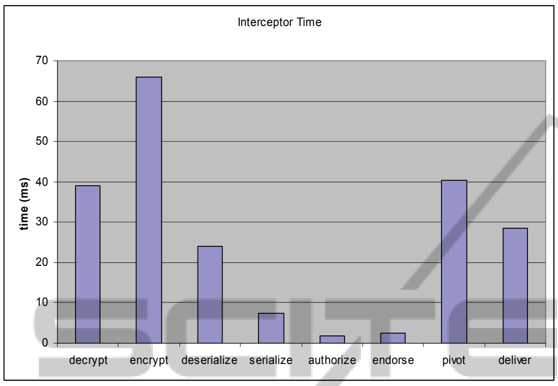

Figure 6: Comparisons of mean interceptor times.

7 CONCLUSIONS

The contributions of this paper are summarized as follows:

- We proposed and developed a lightweight protocol, Compact HTTP, consisting of a small subset of HTTP 1.1, to address message exchange patterns to enable collaborative endpoint network of mobile phones and related applications.
- We developed a REST based framework in the Compact HTTP protocol to handle multi-step interactions, including event subscription and notification.
- We described how to bind HTTP to XMPP and SMS for collaborative mobile phone endpoint network.
- We introduced the concept of hyperlink presence in our approach of HTTP over XMPP.
- We proposed an approach and implemented a solution based on the hyperlink presence in collaborative mobile phone endpoint network to address the broken link problem.
- We described a symmetric key based security protocol in collaborative mobile phone endpoint network to provide end-to-end message level security for service authentication and authorization;
- We developed a prototype system that allows enterprise clients to control and monitor over a dozen of REST resources on a T-Mobile G1 phone.

Experimental studies were performed and our results demonstrated the proposed approaches and architecture are feasible, efficient, and extensible. The future work will be focused on lightweight hypertext representations and application development based on the REST services.

REFERENCES

AlShahwan, F., Moessner, K., 2010. Providing SOAP Web Services and REST Web Services from Mobile Hosts, *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on,* vol., no., pp.174-179, 9-15 May 2010.

Antila, V., Mantyjarvi, J., 2009. Distributed REST Web Services for Mobile Person-to-Person Collaboration, *Next Generation Mobile Applications, Services and Technologies, 2009. NGMAST '09. Third International Conference on*, vol., no., pp.119-124, 15-18 Sept. 2009.

McFaddin, S., Coffman, D., Han, J. H., Jang, H. K., Kim, J. H., Lee, J. K., Lee, M. C., Moon, Y. S.; Narayanaswami, C., Paik, Y. S., Park, J. W., Soroker, D., 2008. Modelling and Managing Mobile Commerce Spaces Using REST Data Services, *Mobile Data Management, 2008. MDM '08. 9th International Conference on*, vol., no., pp.81-89, 27-30 April 2008.

Aijaz, F., Ali, S. Z., Chaudhary, M. A., Walke, B.; 2009. Enabling High Performance Mobile Web Services Provisioning, *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th*, vol., no., pp.1-6, 20-23 Sept. 2009.

Pruter, S.; Golatowski, F.; Timmermann, D.; 2009. Adaptation of resource-oriented service technologies for industrial informatics, *Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE*, vol., no., pp.2399-2404, 3-5 Nov. 2009.

Yong Liu, Connelly, K., 2008. Realizing an Open Ubiquitous Environment in a REST Way, *Web Services, 2008. ICWS '08. IEEE International Conference on*, vol., no., pp.96-103, 23-26 Sept. 2008.

Lozano, D., Galindo, L. A., Garcia, L., 2008. WIMS 2.0: Converging IMS and Web 2.0. Designing REST APIs for the Exposure of Session-Based IMS Capabilities, *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, vol., no., pp.18-24, 16-19 Sept. 2008.

Vlad Stirbu 2010. A REST architecture for adaptive and multi-device application sharing, *Proceedings of the First International Workshop on REST Design*, pp. 62-66, 2010.

Core 2010. *Constrained REST Environments (core)*, https://datatracker.ietf.org/wg/core/.

SOAP 2007. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation* 27 April 2007.

Fielding, Roy, 2000. *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
XMPP 2004. *Extensible Messaging and Presence Protocol (XMPP): Core*, http://tools.ietf.org/html/rfc3920.
HTML 1999. *HTML 4.01 Specification, W3C Recommendation 24 December 1999*, http://www.w3.org/TR/REC-html40/.
Richardson, L.; Ruby, S.; 2007. *REST Web Services*, O'Reilly, 2007.
Li Li; Wu Chou; 2010. Design Patterns for REST Communication Web Services, *ICWS 2010*, pages 512-519, Miami, 5-10 July 2010.
WSDL 2001. *Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001*, http://www.w3.org/TR/wsdl.
SOAP/XMPP 2005. *XEP-0072: SOAP Over XMPP*, http://xmpp.org/extensions/xep-0072.html.
SOAP/JMS 2009. *SOAP over Java Message Service 1.0, W3C Candidate Recommendation* 4 June 2009, http://www.w3.org/TR/soapjms/
Srinivas, Davanum. 2008. http://davanum.wordpress.com/2008/12/29/updated-xmpp-client-for-android/.
Smack 3.1.0 API. http://www.igniterealtime.org/projects/smack/.
SMS 2010. *3GPP TS 23.040 Technical realization of the Short Message Service (SMS) (Release 9)*. http://www.3gpp.org/ftp/Specs/archive/23_series/23.040/23040-930.zip.
RFC 3986. *Uniform Resource Identifier (URI): Generic Syntax*, http://tools.ietf.org/html/rfc3986, January 2005.
WAP 2001. *Wireless Application Protocol*, http://www.openmobilealliance.org/Technical/wapindex.aspx.
WML 2001. *Wireless Markup Language, Version 2.0*, http://www.openmobilealliance.org/tech/affiliates/wap/wap-238-wml-20010911-a.pdf, 11 September 2001.
WSP 2001. *Wireless Session Protocol*, http://www.openmobilealliance.org/tech/affiliates/wap/wap-230-wsp-20010705-a.pdf, 5 July 2001.