Linking Sensor Data to a Cloud-based Storage Server Using a Smartphone Bridge

David D. Rowlands[a], Jason R. Ride[a], Mitchell W. McCarthy[a], Liisa Laakso[b], Daniel A. James[a]
[a]Centre for Wireless Monitoring and Applications, Nathan campus
[b]Applied Physiotherapy and Exercise Science, Gold Coast campus
Griffith University
Brisbane, Queensland, Australia
d.rowlands@griffith.edu.au, j.ride@griffith.edu.au, mitchell.mccarthy@griffithuni.edu.au,
l.laakso@griffith.edu.au, d.james@griffith.edu.au

Abstract – **There is a need for systems that unify the processes of collecting, storing and analysing long-term sensor data in multi-user contexts. The ubiquity and communication features of smartphones make them suitable as data aggregation platforms for standalone sensors. Cloud-based servers are advantageous for their centralisation and scalability in the storage and processing of data. This paper conceptualises a method to link sensors with a cloud server by detailing a smartphone application design (for collecting, aggregating and transmitting data from connected sensors), communication protocol (to preserve security and integrity of data during transmission) and storage methodology (to protect data during processing and storage within the cloud server). Overall, this paper presents design concepts to link sensors with cloud-based storage and provides details of a health-based implementation that applies them.**

Keywords – sensor; smartphone; security; cloud; data.

I. INTRODUCTION

The development of ultra-low power transmission protocols for wireless sensor devices has led to an increase in the number of physiological measurement systems that typify Body Area Networks (BAN) [1][2]. These networks typically collect physiological information from multiple sensors and transmit it to a central receiver for storage, analysis and/or feedback. The process of linking many sensors to a central server can be performed by an aggregation platform tasked with consolidating and securely re-transmitting all data collected from a single body. Current generation smartphones have shown to be suitable for this purpose, in both short-term single-user feedback [3] and longer-term multi-user feedback [4][5] contexts. However, many of the existing multi-user systems tend to focus on direct distribution of data (without analysis) [4] or are tailored to specific targets and lack versatility [5]. There is a need for a more generalised system that integrates collection, aggregation, secure storage and analysis with the potential to incorporate future data formats and processing methods by expanding on rather than replacing the fundamental system design.

This paper outlines the proposed design of a system that uses a smartphone as an aggregation platform to link standalone sensors with a cloud-based server. The cloud server builds on previous work by the authors: a server platform designed to closely integrate storage and analysis of data within a scalable distributed architecture [6]. Section II outlines important design factors in using a smartphone as a sensor bridge. Section III details a practicable strategy for securely and reliably transmitting data to the cloud server, and the methodology employed within the server to protect data during processing and storage. Section IV describes the preliminary development track for a health-based implementation based around the proposed techniques.

II. SMARTPHONE BRIDGE

Traditional methods of collecting data from external sensors such as pedometers that relied on manual user input have been shown to require intrusive "compliance-enhancing features" to sustain a high level of data entry and integrity [7]. As a promising alternative, smartphones can facilitate data collection from sensors by acting as an automated aggregation, storage and communication platform. This allows them to operate as a standalone aggregation platform that provides basic analysis and feedback of data to the user, a bridging device that transfers the data directly to an external system, or a combination of the two. The concept of a bridging device is particularly useful when large amounts of data are being collected, either from one or many users, or there is computationally heavy analysis to be performed that would benefit from the use of dedicated tools.

A. Application Design

When designing a persistent application such as a sensor bridge, the developer must consider that smartphones are capable of performing many concurrent tasks that may frequently and randomly require control of the user interface. Fortunately, smartphone operating systems (OSs) support background states where an application may operate almost invisible to the user without disrupting other applications. This is an ideal method for implementing a sensor bridge, as the majority of its lifecycle is spent collecting data from external devices. In using this method, consideration of user notification, power consumption, and server connectivity factors is essential.

B. User Interaction

While it was mentioned that user reliance in data collection resulted in poor compliance, automated methods may occasionally require some administration in order to operate correctly. A sensor collection application, for

 ISBN: 978-1-61208-207-3

instance, will require a means of modifying variable settings, such as a list of sensors, remote server address or user credentials. There are a variety of methods to achieve such tasks, such as a downloaded configuration file or a simple interface that appears when either the application is installed.

C. User Notification

In the absence of a graphical interface, notification tools such as icons and pop-up messages provide a mechanism of either reassuring the user that the service is operating as intended or to alert them to issues that require their attention. In the context of a sensor bridge, regular events such as sensor status updates and successful data transfers to a remote system do not require user involvement, merely notifying that the system is operating correctly. Non-intrusive, non-blocking methods (e.g., Android's Toast notifications) that simply appear on the screen before fading after a short time are suitable for these events.

Critical events such as sensor or communications hardware failure that may result in data loss demand immediate attention. Therefore, intrusive methods incorporating features such as vibration, audio tones and interface-blocking visual alerts requiring acknowledgement provide the best chance of notifying the user.

D. Power Management

Background applications typically do not require a user interface; therefore power consumption caused by screen lighting is largely insignificant. However, the constant use of communications hardware for a sensor bridge will have a noticeable impact on battery life. To alleviate this, methods are required that help to minimize power use where possible.

It is well established that location tracking using Global Positioning Systems (GPS) can consume considerable power [8] and have a high Time To First Fix (TTFF), making it impractical in applications where location data is requested infrequently. Fortunately, OSs allow location tracking to be achieved without the use of GPS, instead using the known coordinates of cell towers and wireless access points (WiFi) to triangulate the user's position. This can be achieved in a fraction of the time, with a reduced accuracy depending on the availability of network infrastructure and with a significant reduction in power consumption.

Smartphones typically have an idle state of activity where the internal processor enters a low power mode until a wake up event occurs. However, the use of a background application for constant sensor communication will prevent the OS from entering this state, regardless of phone or user activity. By monitoring the incoming data, it may be possible to detect periods of user inactivity where collection can be temporarily suspended (e.g., during sleep).

E. Server Connectivity

A simple implementation of a bridging application may involve a persistent connection with an external system with collected data streamed immediately across. However, long-term collection from multiple sensor devices can generate significant amounts of data that may consume the limited mobile phone data allowance available to the user. Use of WiFi can aid this situation by only uploading when a network connection is available and reserving the cellular network for emergency use. Applying a method such as this will involve the use of temporary storage of data on the mobile phone, so storage capacity must be considered when deciding how long data may be allowed to accumulate before utilizing the cellular network.

In order to ensure that data transfer to an external system maintains a high level of security and integrity, a robust communications protocol must be devised. The following section proposes a protocol that meets these requirements.

III. CENTRALISED SERVER

The data storage methodology used in this system employs and extends an existing system that integrates the collection, storage, analysis and visualisation of multi-channel multi-sensor data within a cloud context [6]. The key elements explored in this section include securely communicating sensor data from multiple mobile clients to a central server, as well as handling reception, manipulation, storage and protection of data within the server.

A. Authentication and Communication

In this system, each connection from client to server represents a link between an identifiable user (with a unique username and secret password) and an arbiter within the server that controls access to the storage and processing tools. The functionality provided by the server is intended for use with private data; this means that all connections must be authenticated so that eavesdropping, modifying or otherwise intercepting communication is not foreseeably possible. Protocols such as HTTPS [9] and WEP/WPA2 (WiFi security) provide some level of protection, but neither have been proven to be infallible [10][11] and are conceptually vulnerable to man-in-the-middle style attacks.

The authentication procedure used in this system aims to prevent man-in-the-middle attacks without the need for a trusted certification authority by using a private key that is never communicated over a network. Figure 1 shows a proposed methodology that implements this authentication procedure, consisting of three distinct parts – login (A), data transfer/processing (B) and logout (C).

The authentication process begins when the client software receives a unique username and confidential password from the user. The two credentials are encrypted using a hashing function to generate a secret cipher key that is stored internally for later use. When a session is required, the client requests the right to be authenticated (Figure 1 A1), which is reciprocated with a pseudo-random unique

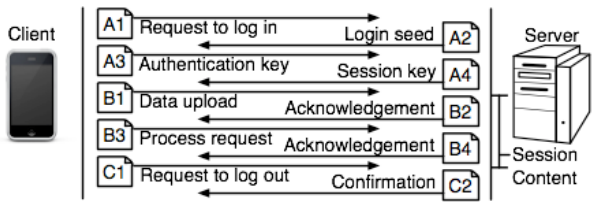

Figure 1. Communication protocol for uploading data

login seed (A2) from the server. The client then uses a second hashing function to produce an authentication key from the secret cipher key and the received login seed. This authentication key (A3) is sent to the server for comparison with an equivalent (generated internally using the same two hashing steps). If the two match, the server replies with a private session key (encrypted using the secret cipher key) (A4) for use on all messages within the session (B1-B4).

Since the authentication key (A3) is dependent on the login seed that preceded it, it cannot be re-used to authenticate illegitimately. By limiting the frequency of login requests (A1), the risk of dictionary-based attacks can be minimised. The length of the session key (A4) controls the relative probability of brute force decryption [11], and is based upon the maximum number of packets expected within any given session. Because the session key is private, symmetric-key encryption techniques such as Integrity Aware Cipher Block Chaining (IACBC) can be used; this particular technique combines confidentiality and verification of correctness into a single computationally efficient algorithm [12]. Replay attacks can be prevented within the server.

Messages sent within the context of an authenticated session may include, but are not limited to, data uploads (B1) and requests to run data processing scripts (B3), both of which are reciprocated with an acknowledgement of success (B2, B4). In data uploads, specific details such as packet size, format, timing and error handling are defined at the discretion of each client, but each upload produces a single data file on the server. Data processing involves collating these packets, optionally synchronising and filtering the content and storing the result. Such requests may occur until the client logs out (C1, C2) or the session is terminated.

This protocol will be further analysed in a future validation study, with the null hypothesis being that socially engineered attacks and weaknesses in the smartphone OS represent the greatest threat to the security of the system. The performance implications have not yet been analysed.

B. Data Reception

Figure 2 details the internal modules within the server relating to reception, processing and storage in the context of uploading data from external clients. Requests from clients are managed by the Input/Output (I/O) handler, which communicates with the File Handler, Script Handler and Session Handler as necessary, depending on the type of request received. The Session Handler also communicates authentication details to the Rule Parser, which protects the Storage Database from unauthorised access.

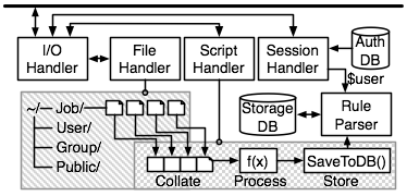

Figure 2. Server modules for receiving, processing and storing data

Each data packet that is uploaded to the server is stored as a single unencrypted file within a virtual scratch space. This space is generated by the File Handler and consists of symbolic links to folders within the file system that an authenticated session has read/write access to, with access to all other folders disabled. These folders include, but are not limited to, private storage (User/), shared storage belonging to the user's group (Group/), a public folder readable by any entity (Public/), and a cache folder representing the context in which actions triggered by communication-based requests takes place (Job/). Data packets would usually be uploaded to the *Job/* folder.

In this system, file-based storage is only intended for short-term use, with permanent data storage handled by a Structured Query Language (SQL) database system. Use of query-based systems such as this shifts the responsibility of identifying, evaluating and collating the requested data away from the user level to the database system framework. In doing so, the system can rely on well-established data acquisition techniques used within SQL frameworks to optimise performance. This structure also allows security features to be implemented by intercepting and manipulating query syntax, as is performed within the Rule Parser.

C. Data Processing and Storage

The transition from temporary file storage to long-term database storage is achieved through a process of collating the uploaded data packets into a contiguous data set, processing and filtering the data as necessary, and storing the result into the database. The exact procedure is context dependent and intended to be versatile, and as such, is suited to a scripting environment such as MATLAB [6]. For protection of the server [13], the code execution environment is sandboxed, with resource access controlled by the Script Handler. In scripts used to upload data, provisional access is given to files within the Job/ folder for input and the Storage Database (via the Rule Parser) for output.

D. Data Protection

The purpose of the Rule Parser is to prevent unauthorised access to data by validating and amending the syntax of all SQL queries before they reach the Storage Database. The current implementation focuses on asserting a straightforward permission directive – the content of a table row may only be updated by the same user who created it, although other users may also be given permission to read it. To enforce this, all tables within the SQL database contain two hidden fields (_owner and _readers), which are used to determine the access rights of the currently authenticated user ($user) specified by the Session Handler (see Figure 2).

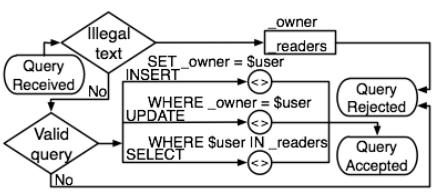

Figure 3. Query evaluation flowchart; simplified Rule Parser

A simplified representation of the algorithm used to enforce access rights is shown in Figure 3. Upon receiving a query, the Rule Parser verifies that the query text contains no mention of the words *_owner* or *_readers*, since either could constitute an attempt to manipulate or otherwise access hidden fields. If either keyword is found, the query is rejected. The query is also rejected if it is not an attempt to *INSERT* (create), *UPDATE* (modify) or *SELECT* (read) one or more rows. In the case of an *INSERT* statement, the query text is modified to define the *_owner* as being the *$user*. In an *UPDATE* statement, the query text is modified to ensure that *$user* is the *_owner* in all rows to be updated. In a *SELECT* statement, the query text is modified to ensure that *$user* exists within a list of *_readers*. It is noted that the full implementation will comprehensively evaluate syntax (including wildcards) to prevent misuse and provide functionality to add and remove users from the *_readers* list.

IV. CURRENT DEVELOPMENT

A health-based application is being developed using a smartphone (running Android OS) containing WiFi, location tracking, Bluetooth and an ANT receiver for communication with compatible wireless sensors. The initial development of this application will involve acquisition of data using an ANT compatible footpod. The sensor determines stride, cadence and distance using a triaxial accelerometer and transmits this information to the phone at a rate of 2Hz.

The application is implemented as a background service that commences during the boot sequence of the OS. This service initialises the ANT communications channel with the sensor and stores the received data into a temporary file. Location tracking provided by the phone is also utilised to obtain geographical position intermittently, with the option of GPS or network services depending on application requirements. In this system, network services are heavily favoured over GPS, trading off reduced accuracy for significantly reduced power consumption and time to first fix, as the hardware is only required to be active long enough to triangulate the user's position before switching off. The use of GPS allows its impact on battery life to be measured, as well as providing a relative measurement of accuracy.

Data upload to the server occurs based on the availability of WiFi and 3G networks. While a WiFi connection is present, data latency is kept low through frequent uploads. If WiFi is unavailable, data is accumulated in local storage until a connection becomes available. Once a local storage limit is reached, accumulated data is uploaded using the 3G network. If neither network is available, there is little choice but to notify the user and request immediate action.

Due to the dependence on network availability, data packets uploaded to the server may be unpredictable in size and content. At least once a day, the client will request that a script be run to collate, process and store all data currently held within uploaded packets. To do this, the script creates a contiguous stream of time domain samples, with each sample stored as an independent entry into the database. Each sample has an owner, timestamp, source (e.g., ANT or GPS) and a set of numeric fields used to describe sample content. An ANT sample will contain stride, cadence and distance, while a GPS sample will contain latitude, longitude, accuracy and time to first fix. In future development, the cloud server will process this data to produce useful statistics to users and their supervising practitioners.

V. CONCLUSIONS

This paper has outlined the design of a sensor bridge application and a communication and storage protocol to allow long-term multi-user sensor data to be securely uploaded to a remote centralised server. Further work will involve a validation of the prototype implementation to expand the capabilities of both the data collection and analysis aspects of the system. Due to the generalised design of the overall system, it can be applied to various real world applications in areas such as health and sport.

REFERENCES

[1] C. Otto, A. Milenkovic, C. Sanders, and E. Jovanov. "System architecture of a wireless body area sensor network for ubiquitous health monitoring", Journal of Mobile Multimedia, 1(4):307--326, 2006.

[2] E. Jovanov, K. Frith, F. Anderson, M. Milosevic and M. C. Shrove. "Real-time Monitoring of Occupational Stress of Nurses", Proc. of the 33rd Annual Conf. of the IEEE Eng. in Medicine and Biology Society 2011;2011:3640-3

[3] T. McNab, D. A. James, D. Rowlands, "iPhone sensor platforms: Applications to sports monitoring," Procedia Engineering, Volume 13, 2011, pp. 507-512

[4] C.L. Borgman, J. C. Wallis, M. S. Mayernik, A. Pepe. "Drowning in data: digital library architecture to support scientific use of embedded sensor networks", Proc. of the 7th ACM/IEEE JCDL, 2007, 269-277

[5] A. Bourouis, M. Feham and A. Bouchachia, "Ubiquitous Mobile Health Monitoring System for elderly (UMHMSE)", International Journal of Computer Science & Information Technology (IJCSIT), Volume 3, Issue 3, June 2011.

[6] J. R. Ride, D. A. James, J. B. Lee and D. D. Rowlands, "A distributed architecture for storing and processing multi-channel multi-sensor athlete performance data," Proc. of the 9th Conf. of the Intl. Sports Engineering Association, in press.

[7] A. A. Stone, S. Shiffman, J. E. Schwartz, J. E. Broderick and Michael R Hufford, "Patient compliance with paper and electronic diaries," Controlled Clinical Trials, Volume 24, Issue 2, April 2003, Pages 182-199

[8] Aaron Carroll , Gernot Heiser, An analysis of power consumption in a smartphone, Proceedings of the 2010 USENIX conference on USENIX annual technical conference, p.21-21, June 23-25, 2010, Boston, MA

[9] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," IETF RFC 2246, 1999.

[10] F. Callegati, W. Cerroni and M. Ramilli, "Man-in-the-Middle Attack to the HTTPS Protocol," IEEE Security & Privacy, IEEE, 2009, pp. 78-81.

[11] V. Kumkar, A. Tiwari, P. Tiwari, A. Gupta and S. Shrawne, "Vulnerabilities of Wireless Security protocols (WEP and WPA2)", Intl. Journal of Advanced Research in Computer Engineering & Technology, Volume 1, Issue 2, April 2012.

[12] C. Jutla, "Encryption Modes with Almost Free Message Integrity," Advances in Cryptology - EUROCRYPT 2001, B. Pfitzmann (ed.), LNCS 2045, Springer Verlag, 2001.

[13] K. Gama and D. Donsez, "A self-healing component sandbox for untrustworthy third party code execution," Proc. of the 13th Intl. Symp. on Component-Based Software Engineering, Springer, 2010, pp. 130–149.

 ISBN: 978-1-61208-207-3