RoboTractor: A Secure Signaling and Control System for Remote Management of Agricultural Vehicles using XMPP and REST

Jeremy Wright
Arizona State University
jlwrigh1@asu.edu

Arun Balaji Buduru
Arizona State University
abuduru@asu.edu

David Lucero
Arizona State University
dwlucero@asu.edu

***Abstract*—The need for the automation of vehicles is increasing requiring improved efficiency, operation, and reduced operational costs. The scope of this project is to build a signaling and control system for automation of vehicles, with a focus on security to make the systems safer by drastically mitigating the risks of the systems from being compromised. An attack against these systems could allow for the vehicles to be stolen, damaged, hijacked, or cause other losses. The main tasks in this project are to build a secure front-end web interface for users to give instructions and view vehicle status – built upon a Representational State Transfer (REST) architecture, set up an Extensible Messaging and Presence Protocol (XMPP) server and client with required authentication, data integrity check mechanisms and encryption, and build a simulated automated vehicle with simulated Trusted Platform Module (TPM) similar to what would be used to secure a real vehicle, and to enable the simulated vehicle to be capable of interfacing with the XMPP components and front-end.**

***Index Terms*—Secure Signaling and Control, Remote Management, Vehicle Automation, Self-Driven Vehicles, TPM, Root of Trust, Trusted Platform, TrustZone, ARM, Machine Automation, XMPP, Rest, RESTful methodology, Simulation, Framework, Cryptography**

I. Introduction

Agriculture has a growing problem. Crop input costs such as water, fertilizer, pesticides, and fuel, are increasing. The global population is increasing, driving the need for more food. Local, State, and Federal governments are increasing oversight and reporting requirements on growers. Land area available for planting, and harvesting is decreasing globally. All this pushes farmers to rely on technology to improve their efficiency while reducing recurrent costs. Take government oversight requirements [1], growers are required to track where, and how much material is spread into the fields. This can be a considerable overhead for growers, however an automated machine can do this quite easily, and in real-time if desired. This project demonstrates how to securely connect an agricultural machine to an automation server in a secure manner using existing internet technologies. Using existing technologies is a critical point. The market is already stretched thin, and incurring new infrastructure in rural areas can be prohibitively expensive, however leveraging the existing Internet infrastructure, we can provide high levels of service in a secure manner.

The most important requirement in developing a secure signal and control system is ensuring authentication of vehicles and data integrity, and securing communication channels. The path plan and location feedback transmission needs to be secured and protected from eavesdroppers, preventing attackers from compromising and/or stealing these vehicles. In this project we use the Django web framework to handle the front end interface and server duties. Django facilitates the use of python libraries and components to interface between our Front-end, server, and client-side vehicle simulators. The end goal of RoboTractor is to have a comprehensive tool that translates the abstract user inputs into remotely executable actions for agricultural vehicles in a secure manner using advanced web, network, and cryptographic techniques.

RoboTractor is a framework to provide integrity, authentication, and root of trust services for machine automation specifically within an agriculture domain. Electronic Control Unit (ECU) manufacturers

will particularly be interested in the root-of-trust details described in Section III-B. Agricultural service providers have typically been subject to locality issues. Historically, it's been difficult to provide agricultural services at a reasonable cost, at any appreciable distance from the farm itself. RoboTractor provides a conduit for service providers to provide their services at a distance. This allows expert agronomists to consult with farms at any distance. Not only can this reduce cost, but it also allows growers access to a marketplace of services before unavailable to them. RoboTractor is the foundation for connecting the needs of the growers to the providers.

II. System Models

A. System Model

RoboTractor leverages the Django Web framework [2] to realize the required interfaces. Figure 1 describes the connection of these components. The combination of XMPP and REST in RoboTractor is a demonstration of how to extend the existing HTTP development environment i.e. "the web of things" into a stateful protocol. HTTP is by design stateless. XMPP on the other hand is a stateful streaming connection between two clients. In this case the vehicle (tractor simulation) and a command server. To achieve this mesh we will leverage a custom XMPP script to bridge gap between Django's RESTful interface and the streaming XMPP protocol securely and efficiently over a secure HTTP connection. This is essential for a scalable web service.

B. Software

1) XMPP Server: XMPP stands for Extensible Messaging and Presence Protocol. XMPP is a communication protocol that relies upon XML (Extensible Markup Language) for operation. XMPP is an open standard that operates with a client-server architecture. Anyone may design, build, or operate their own server and/or client, and be able to communicate with others operating under the open standard. This means that XMPP is strongly decentralized, with no master or control servers. Additionally, XMPP has support for security protocols such as TLS built into its base specification. XMPP operates by opening an XML document or session between two peers. Within this session, xml stanzas can be sent containing information. RoboTractor uses the ejabberd [3] XMPP server as it provides an existing Python interface, to integrate with the rest of our Python based ecosystem. The vehicle simulators themselves use the open source SleekXMPP Python XMPP library as the client side XMPP interface. This allows communication between the vehicle simulators and the ejabberd XMPP server.

2) REST Interface: REST is a web idiom which stands for REpresentational State Transfer. REST allows a website to expose data via a set of URLs. In this way REST provides a common interface to any HTTP enabled client. As HTTP becomes more and more pervasive REST allows one to aggregate several data sources with simple tools. Robotractor leverages the TastyPie framework to expose the Django models via an idiomatic REST interface [4]. To demonstrate the REST interaction with XMPP, Robotractor uses AJAX (Asynchronous Javascript And XML) queries to interact with the data. This has the added benefit of making the site more responsive without requiring page refreshes to see new data. Essentially, the REST interface POSTs commands to the database. The database acts as a queue holding the messages until the XMPP adapter is ready to deliver them to tractor simulator.

3) Vehicle Simulator: The Vehicle Simulator is the device or machinery that is meant to be the recipient of the secure command controls and signals. For the RoboTractor project, this simulation is entirely custom built in Python. The Simulator operates on a state machine principle, with states for receiving data, operating on data, uploading data, and rest and shutdown. The vehicle simulator is designed to be extensible and easy to modify, with tunable parameters for the vehicle.

C. Security Model

RoboTractor deals with two primary security classes: One involving the vehicles and other assets, and one involving the system components and their communication channels. Because the project is web-based and uses PKI, numerous attack models have been created that this project will deal with. Within the class of communication channels, attacks may be present against the authentication systems of both users and assets. Robotractor currently

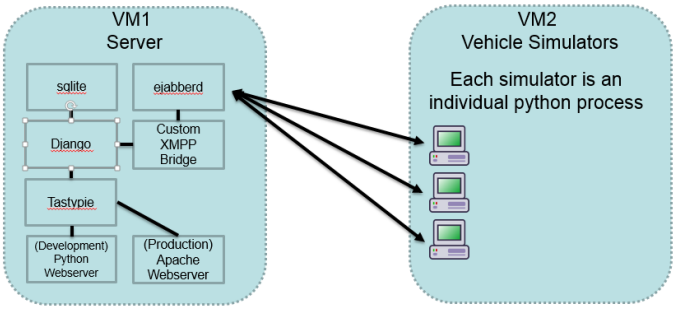


Fig. 1. Software Component Block Diagram

employs advanced two-factor authentication. Usage of TLS and other best common practices will be implemented to prevent against replay attacks on the system, and also for establishing a secure session between assets and control. A dual-encryption scheme has been used to send encrypted control signals within an encrypted XMPP stanza to ensure high information assurance. Because this project utilizes Public Key Infrastructure, key protection is important. To this end the RoboTractor project uses secure keys on assets with a Trusted Platform Module. By making both key-pairs private and secure, RoboTractor provides advanced communication security. On the other end, the server components must be secured as well using best common practices in server defense, firewall, patches, updates, etc. Additionally, the RoboTractor system will implement an advanced GPS location reporting system to provide physical security in case of asset theft. By defending against these attack models, the RoboTractor project presents the realization of an advanced system for signaling and control with high security.

III. Project Description

Implementation is divided into 2 primary phases (Marked as milestones on the attached Gantt chart). In the first integration phase we connected the various libraries and off the shelf components. We made sure that this integration did not weaken, but rather strengthened the security properties provided by each independent component.

In the second phase we extended these components to provide command and control of agricultural vehicles. For schedule mitigation, David and Arun completed this phase before the framework is

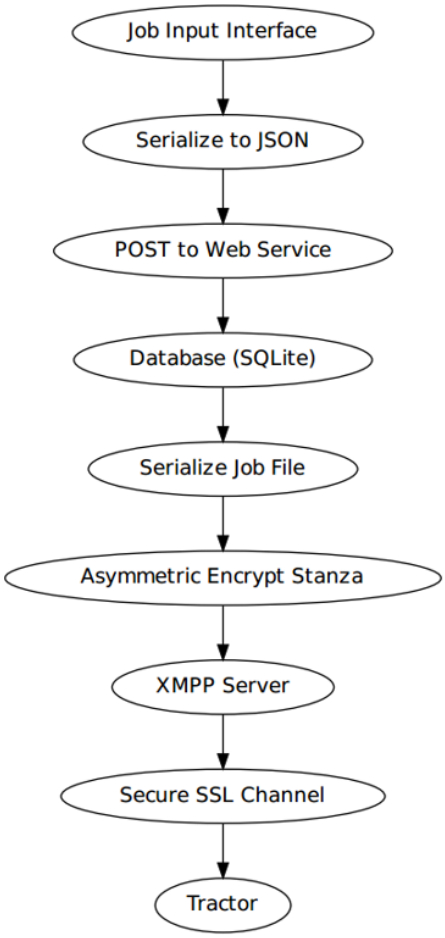


Fig. 2. High Level Signal Path

assembled. Once the framework was in place, David and Arun's interface designs were merged with the overall system. After this merging, development switched focus to testing and bug fixing, since the main components were operating at satisfactory levels. This also allowed refinement of the front-end interface to better suit the needs of both the project, and the rest of the RoboTractor components. As of the creation of this document, all Midterm and Final deliverables have been achieved. The following tasks have been updated to reflect work done, and remaining future improvements that are beyond the scope of this project.

A. Task 1 : Development Environment

This task consisted of setting up the development environment for the RoboTractor project. A project

homepage was provided to us via the ASU virtual lab project. This home page provides source control in the form of a GIT repository which our project takes advantage of. In addition we used the amazon's cloud service to develop and test our project. Additionally, two virtual machines were provided via the ASU virtual lab project to use as our project servers. Setup of these servers was automated with the creation of a script in order to download all required software packages. Remaining work for this task is to enable proper connectivity between the two assigned Virtual Machines. RoboTractor is built upon the Django Web Framework. The development environment is best described in [2]. The Development environment consists of two fabric configurations: one for development and a second for deployment. This allows development settings to be more relaxed facilitating quick coding cycles, while the deployed configuration is more secure. Separate database also prevent test data from polluting the deployed demo project.

B. Task 2: XMPP Implementation

XMPP provides an SSL/TLS connection between the tractor, and the server. This allows a base level of secure communication. However to augment the security model, we provide an additional asymmetric encryption of the command structures in each XMPP stanza. This project also leverages a Trusted Platform Module (TMP). TMP is a hardware module, either part of the compute engine complex or as in ARM processors integrated into the CPU core itself. ARM's offering is called TrustZone. These security services provide a hardened storage location for private keys. One cannot extract the key without destroying the device, and the key itself. These devices also leverage a best practice called "Root of Trust" to protect against known cipher attacks. [5]. The hardware modules only decrypt/encrypt material if the system is in a "secure" state. Typically this is obtained by the compute core booting an internal, untamperable ROM loader. This loaders verifies the second stage bootloader with a cryptographic signature stored in the TPM device at production time. If this signature passes, the TPM is unlocked and the hardware may continue booting. If any signature fails, the TPM remains locked, and refused to cipher information.

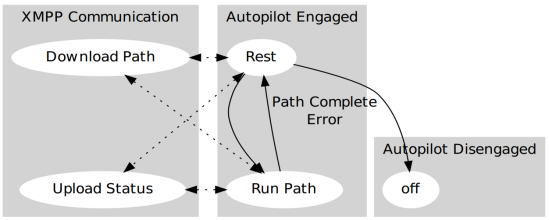


Fig. 3. Tractor Software State Machine

For the purposes of RoboTractor this behavior will be simulated at the API level. However hardware integrators may leverage the best practices laid out here to implement a "root of trust" to include the automation server. Specifically in RoboTractor, this TPM is simulated with an additional Python module entitled TPM.py. By having to include this module on both the server and client sides, the full TPM functionality is simulated. Without this module, the system will not be functional.

C. Task 3: Simulation Tractor Agent

The final deliverable for this task was to implement the tractor simulator. This has been achieved after multiple team discussions, and a state machine based vehicle simulator was written in python. Tasks such as XMPP communication, path processing, and status reporting were completed, and the simulator was updated with its required functionality. The state machine consumes the Job File format described in Figure 4. Additionally, the vehicle simulator employs usage of public-key cryptography for its communications with the XMPP server. This ensures that the control and signals between the two are secure and provide high information assurance. Upon initial startup of the vehicle simulator agent, a session-key request message is encrypted with the server's public key. Once the server decrypts this request, it is processed, and a session key encrypted with the agent's public key is sent back. The vehicle simulator agent then decrypts the session key. This session key is used from here on out to encrypt and decrypt communication data that is being sent as the "body" of an XMPP message.

Because the goal of the RoboTractor project is to for remote control and signaling of an automated vehicle, mapping and coordinates play a big role in the simulator's operation. As a result, much of the operation of the vehicle simulation agent relies on

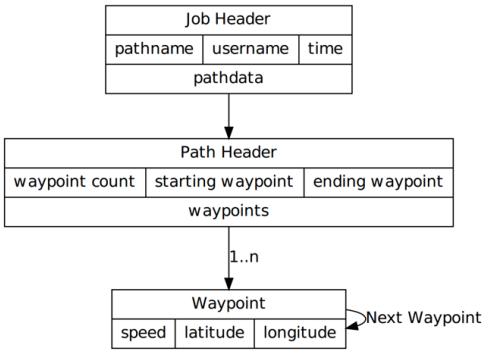


Fig. 4. Job File Format

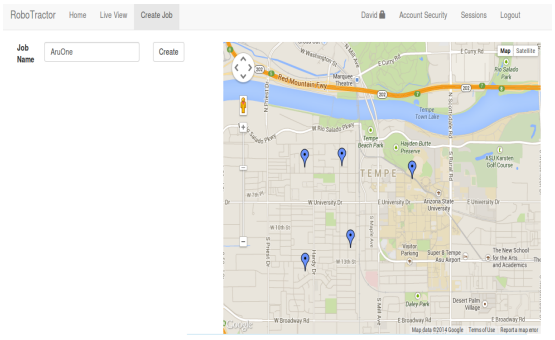


Fig. 5. Job Input Tool

mathematical formulas involving GPS coordinates. Thorough testing of these algorithms and formulas had to be done to ensure that the operation of RoboTractor was as intended. Current operation of the Vehicle Simulation Tractor is as follows: Upon startup, the tractor enters a state in which it queries the XMPP server to provide a session key (Figure 3). Once this session key is securely established, the simulation agent sends an encrypted request for job data. The XMPP Server interfaces with the database and front-end to build a JSON object containing this data, encrypts it, and sends it to the simulation agent. From here, the simulation agent processes the data to build its waypoints and path, and begins its simulated 'movement'. After every tic of movement, the agent sends an encrypted status update back to the XMPP server, which can then update the Front-end for user visibility.

As described in Figure 2, the job files are stored in the database. When the Tractor is instructed to run a job, the Tractor downloads the Job file from the server over its secure XMPP connection. The backend server is responsible for translating the database models into a JSON encoded Job. The tractor simulation agent decodes this JSON encoded job, and then processes its accompanying data fields as needed. Similarly, when sending status and waypoint data back to the server, it is first encoded as a JSON object, then encrypted, then sent. File [6].

D. Task 4: Frontend UI

The "single-page" web application is the modern design methodology to web applications today. Leveraging this design architecture the front-end queries the REST API to draw the position of all tractors within a Google Map context. Additionally, the frontend user interface is to be highly secure. Since the web frontend this provides a control channel to functioning machinery, security directly correlates to the safety of the system. This task has three high level sub tasks:

1) Path Design and Input
2) Map Rendering
3) Two-Factor Authentication

Via the web interface the user is able to input a path for a given tractor to drive (Figure 5). Our map rendering system allows the user to select the waypoints just by clicking on the map for which the system returns the latitude and longitude location information to the REST interface, once successfully posted (HTTP 201), the UI renders the respective point on the map. The user then can select that location as a waypoint, on top of which the real-time markers will be placed to display location and waypoint information. Through this method the user can select a number of waypoints and generate the path for a tractor. The path generation algorithm generates the path by drawing a straight line between the waypoints in the order of the waypoints selected by the user. The generated path is then laid on top of the rendered Google map. By this way the user can view the path on the real time map.

The webservice stores the waypoints in a Model format until instructed to convert it to the Job Format and send it to the Tractor. The Tractor receives this path over the XMPP channel through the aforementioned secure communication channels.

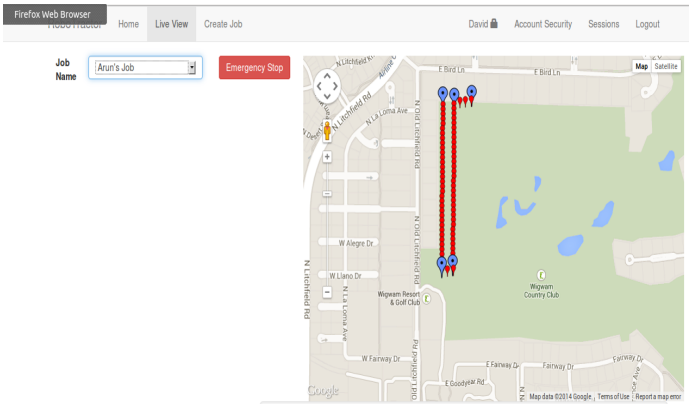


Fig. 6. Live Job View Page

The UI periodically queries REST endpoint for the position updates since the last query, which the XMPP server writes to a database after it receives position updates from the Vehicle simulation agent, and updates it on the Google Map. Additionally, robotractor has a real-time path view functionality where the farmer/owner can view the agricultural vehicle status in real-time. The vehicle is designed to report its location every five seconds. The location of all the vehicles is updated to the database and then live view functionality pulls the latitude and longitude location information from the database and renders them on the Google map. This interface as shown in Figure 6. This function also has security implications as the location tracking can also mitigate vehicular theft. Since all the communications are protected by a dual layer encryption, the transmissions cannot be tampered with easily. Hence it is difficult for an attacker to perform attacks on the system.

The RoboTractor also have schedule and boundary functionality, where the farmer/owner can use the scheduler functionality to upload job to the database without actually launching it, which can be done later. The Vehicle Location Validation functionality allows the farmer/owner to set up the boundary of his/her farm beyond which the vehicles will be either made in-operable or disabled to mitigate the vehicular thefts or accidents. Within the Vehicle Simulator designed this is as simple as moving the vehicle to the shutdown state.

E. Task 5: Server Backend

The Server Backend is the critical component which connects all further subsystems. Extensive knowledge of Django will make this task easier. As it will require linking multiple components together in a orchestrated fashion. Our initial design used an existing Punjab library to connect Django to eJabberd [7]. However once we began implementing the interface we found since were were extending the XMPP session only to the REST interface and not all the way to the web browser, the advanced functionality of Punjab was unnecessary. Instead we used SleekXMPP to setup the backend as an XMPP client. All tractors in the system connect to this XMPP endpoint, and the server indexes the Jabber ids against registered machines in the database. REST communicates to the database. This backend service uses that database to communicate to clients.

We have currently setup and tested our server running on Django. This task also includes the Service Side Session-Based Authentication. The Server is responsible for generating the Time based One-Time Password (TOTP) as described in RFC 6238 [8]. TOTP provides the “thing you have” authentication factor. RFC 6238 describes generating a QR code from a private seed value. This setups up the pre-shared key between the user and the web service. At this point the user may use an standard authenticator app, such as Microsoft Authenticator or Google Authenticator. At each sign on, the user is prompted for their password “the thing you know”, as well as a random number from their device, ”the thing you have”. Additionally, this helps prevent replay sign-on attacks since the transaction requires knowledge of the pre-shared key to generate the unique sign on credentials.

F. Task 6: REST Interface

The REST interface serves the primary means of interacting with the system. The Web interface exists as a “single-page” application which leverages AJAX principles over this REST API. The REST interface is currently functional, and will be updated to interface further with the frontend interface and backend components as they are completed.

The REST API is self documenting in the JSON format. Each endpoint publishes the data, as well as its supported services on the

`/schema?format=json` endpoints. While this provides API developers a straightforward way to view the supported data, and required formats, being in JSON, a developer can also dynamically discover the API and generate a site against it. This is a common idiom in enterprise web applications using dependency injection. RoboTractor supports this idiom over REST for more advanced use cases.

Lastly, since the REST interface is a command interface to actual machines, it is critical that all requests are authenticated. The REST interface requires all queries to come from two-factor authenticated sessions (as managed by the backend service). Each transaction contains a secured Cross-Site Forgery Request Token, and all transport is secured by SSL. This is in line with best practices for web services [9].

G. Project Task Allocation

From the tasks outlined in Section III, the breakdown of work is as follows: Jeremy Wright has configured the development environments and configuration and development of the REST and Frontend interface. Jeremy also researched into modeling trust zones which allow us to simulate trusted hardware in actual vehicles. He operated as the project lead due to his experience in Python-based Web Applications. Arun Balaji Buduru worked on the Server Backend and Frontend UI components, specifically with regards to the mapping interface and google maps API. David Lucero worked primarily on the Tractor simulation agents and their XMPP implementation, as well as assisting Jeremy in interfacing components together. Each group member has approximately 30% workload of the project with the remaining 10% shared due to the interconnected nature of all components. A detailed task breakdown can be seen in the attached Timeline.

H. Deliverables

Midterm Deliverables:

1) Functioning REST Interface (completed)
2) Front-end Test Interface (completed, with additional two factor-authentication)
3) API documentation for using the REST interface as an external service. (completed)
4) API documentation for using the XMPP interface to act as a vehicle. (moved to final deliverable)
5) Finalized Design of Vehicle simulator (completed, with additional python test implementation)
6) Interim Progress Report (completed)
7) Updated Project Schedule (completed)
8) Interim Project PowerPoint Presentation (completed)

Final Deliverables:

1) Completed, secure front-end interface including two-factor auth and SSL/TLS (complete)
2) Vehicle Simulator with working component interfaces (complete)
3) Simulated Trust Zone Functionality (complete)
4) Asymmetric XMPP message encryption (complete)
5) Final Project Report (complete)
6) Final Project PowerPoint Presentation (complete)

I. Project Timeline

The attached project timeline (generated from Microsoft Project) describes the overall tasks of the project.

IV. Risk Management of the Project

Several potential issues have been identified that may pose risk to successful completion of this project. These risks have been identified in Table I. Along with the description, ratings have been assigned to each risk identified, and potential mitigation strategies are described. During the course of project development, each of these risks was taken into consideration. For the risk of connections between components, the mitigation strategy was to include security on component connections. In the final version of the RoboTractor project this is done in two ways. The first way is that the XMPP server/client connections use SSL/TLS by default. This means that their communication channel is secured. In addition, the actual messages passed between the simulator and the tractor are themselves encrypted, providing an additional level of advanced security on the communication channel.

Another risk identified was the correctness of the vehicle simulation agents. The mitigation strategy of this risk was to fully design the simulation agents first, before development, and to review and test the design to ensure it would meet all requirements. This strategy was carried out, and the vehicle simulation agent itself is highly extensible. This allows easy changes and modifications if needed, while maintaining functionality and security.

The third risk identified was improper configuration of components, which could lead to entire system functionality breaking down. The mitigation strategy identified for this risk was to thoroughly review and test components before, during, and after development. This strategy was carried out, by repeatedly testing the components as they were begin developed, with actual geographic waypoint data that the system would use if run in real life. Using this actual data also allowed us to verify the the algorithms and formulas used were operating correctly.

The fourth risk identified was that of using incorrectly implementing PKI or configuring it incorrectly. This is a common risk, and is hard to detect. Our mitigation strategy was to keep it simple and follow best practices and methodologies. To accomplish this, we limited the cryptographic use in RoboTractor to only what was needed, and used standardized algorithms implemented with a widely accepted library (Pycrypto).

The last risk identified was that of Service uptime and web reliability. Because the RoboTractor project is web-based, this is very important. If the project is down, this means a farmer or vehicle operator would be unable to operate their machinery remotely. Our mitigation strategy for this risk was redundancy, and using cloud hosting to alleviate hardware reliance. This risk did actually make itself present during RoboTractor development, as the server and code repository we had requested through the ASU cloud system repeatedly went down. Instead, we used a secondary cloud hosting solution temporarily so that development and testing would not be hampered.

V. Conclusion

This work continues to explore how REST and XMPP can help alleviate the growing cost issues in Agriculture. By providing a base set of secure services future integrators can extend the provided data services over this base framework. Using the RoboTractor framework accelerates development by allowing content providers to provide secure, integrity, and authentication to their own agricultural focused service.

Agricultural machines themselves are growing in complexity, and the number of ECUs, integrated sensors, advanced engine management, government oversight documentation, prescription maps, real-time agronomy, crop consulting, genetic seeding, the list is unbounded. These services can all be accelerated, reduced cost, and increased quality and timeliness with the addition of secure connectivity. RoboTractor provides the framework for these content provides to provide such services. It's an exciting time to see where growers are incorporating technology to bring us our daily bread.

References

[1] N. C. D. of Agriculture & Consumer Services. (2013). Growers now have fertilizer, diesel cost information available | cotton content from southeast farm press, [Online]. Available: http://southeastfarmpress.com/cotton/north-carolina-report-will-track-fertilizer-diesel-costs-growers (visited on 03/08/2014).

[2] A. Holovaty. (2014). Django, GitHub, [Online]. Available: https://github.com/django (visited on 02/06/2014).

[3] A. Shchepin. (2006). Ejabberd community site | the erlang Jabber/XMPP daemon, [Online]. Available: http://www.ejabberd.im/ (visited on 02/06/2014).

[4] D. Lindsley. (2014). Toastdriven/django-tastypie, GitHub, [Online]. Available: https://github.com/toastdriven/django-tastypie (visited on 02/06/2014).

[5] G. Shpantzer. (Jun. 2013). Implementing hardware roots of trust, [Online]. Available: http://www.trustedcomputinggroup.org/files/temp/76882F9C-1A4B-B294-D09D38B918AD23D0/SANS%20Implementing%20Hardware%20Roots%20of%20Trust.pdf (visited on 03/09/2014).

[6] (2014). JSON, [Online]. Available: http://www.json.org/ (visited on 03/09/2014).

[7] C. Zorn. (2014). Twonds/punjab, GitHub, [Online]. Available: https://github.com/twonds/punjab (visited on 02/06/2014).

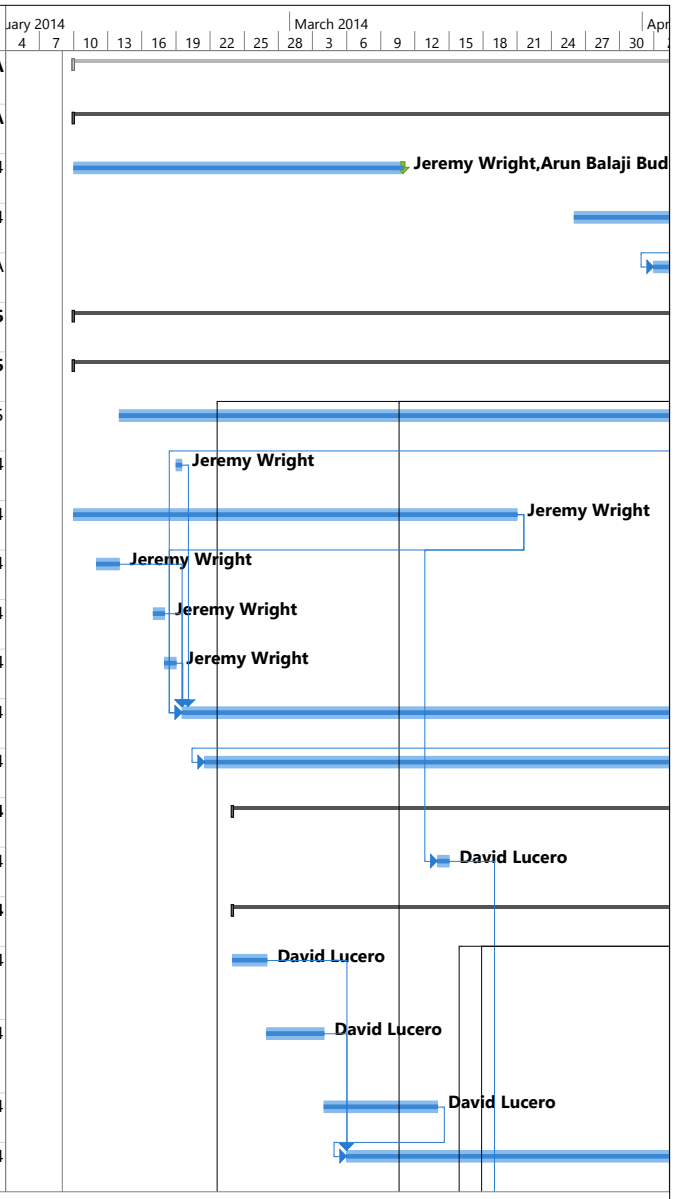
[8] J. Rydell, M. Pei, and S. Machani. (2011). TOTP: time-based one-time password algorithm, [Online]. Available: http://tools.ietf.org/html/rfc6238 (visited on 03/09/2014).

[9] H. Adams. (Oct. 1, 2002). Best practices for web services: part 1, back to the basics, [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-best1/index.html (visited on 03/09/2014).

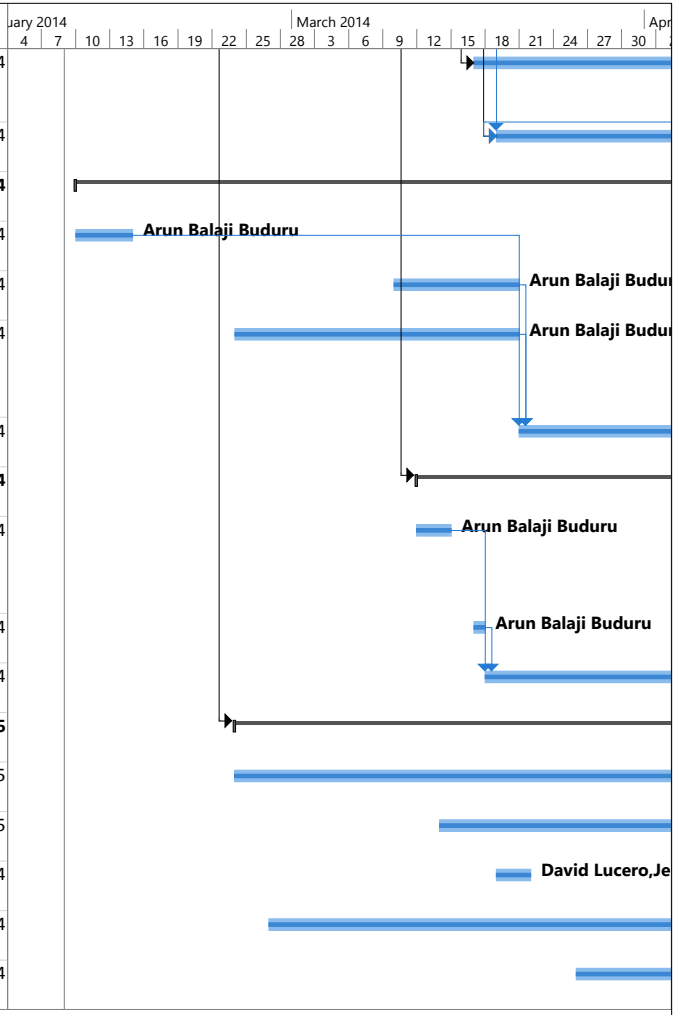
TABLE I
RISK MANAGEMENT

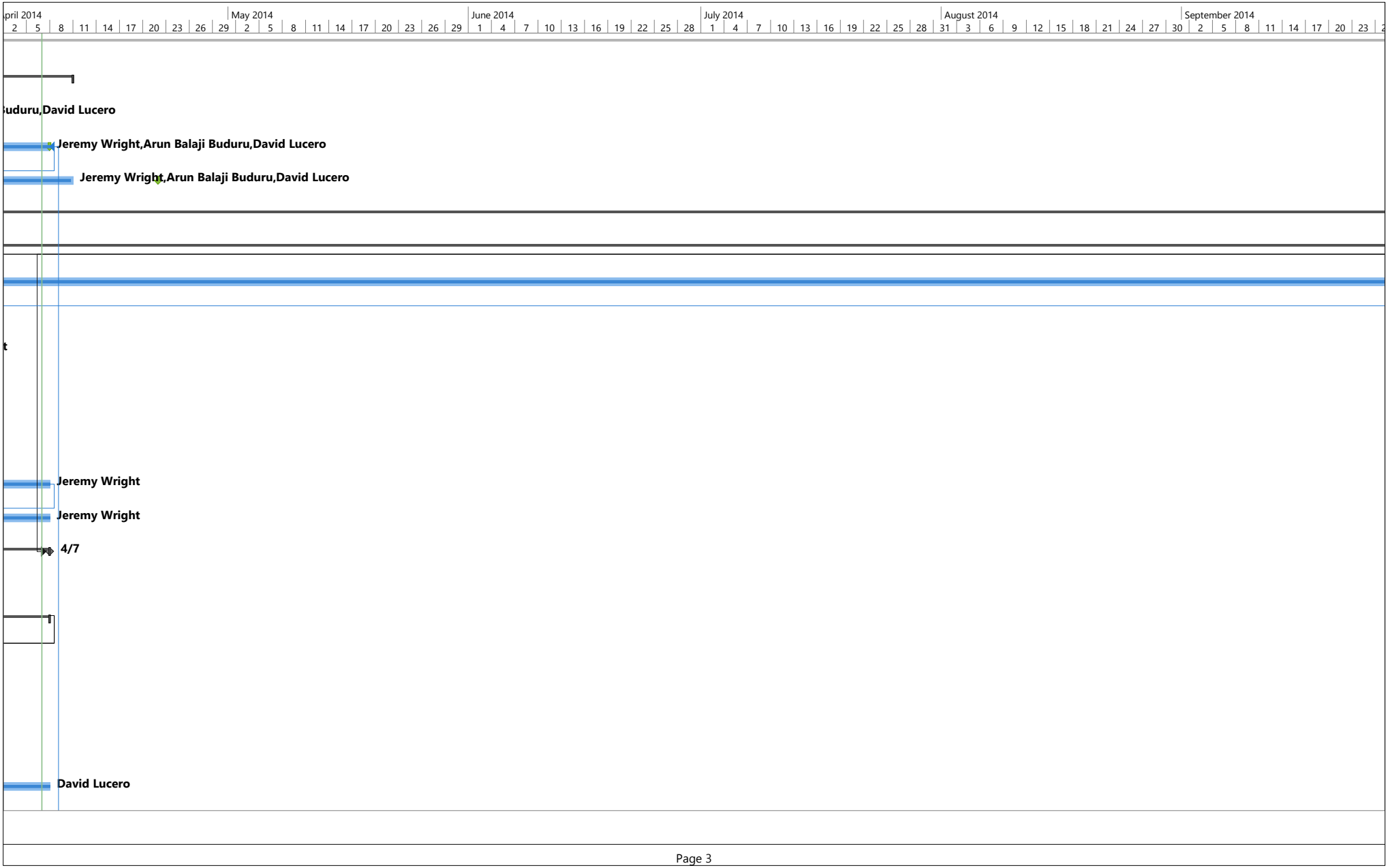
Risk Description	Risk of Failure	Consequence of Failure	Mitigation Strategy
Connections between components must be secure	Low	Product would still function, but would lack security information assurance will suffer	Plan to include security on component connections ahead of time. Follow best practices (SSL/TLS)
Evaluation of Project depends on proper vehicle agents	Medium	System is unable to be tested if vehicle agents are inoperable or incorrect	Vehicle agents will be designed first to ensure compatibility with all other components. Thorough review and testing of component to ensure proper functionality
Improper configuration of components	Medium	Components do not communicate properly with each other. System functionality is void	Thorough reviews and testing of components to ensure proper functionality and communication
Incorrect PKI implementation and/or configuration	Medium	Puts all identified security domains at risk. PKI used becomes useless. System becomes vulnerable to outside attacks i.e. MITM	Follow best practices and standards. No re-invention of the wheel
Service Uptime (web reliably)	High	Product is unusable if network connections are not operable	Redundancy. Cloud hosting to alleviate hardware reliance

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names	% Work Complete	Actual Finish
0		**RoboTractor XMPP and REST Command and Control Server**	**274 days**	**Mon 2/10/14**	**Thu 2/26/15**			**99%**	**NA**
1		**Administrativa**	**44 days**	**Mon 2/10/14**	**Thu 4/10/14**			**99%**	**NA**
2		Interim Report	4.2 wks	Mon 2/10/14	Mon 3/10/14		Jeremy Wright,Ar	100%	Mon 3/10/14
3		Final Project Report	1.8 wks	Wed 3/26/14	Mon 4/7/14	38FF	Jeremy Wright,Arun	100%	Mon 4/7/14
4		Project Demo	7 days	Tue 4/1/14	Thu 4/10/14	3	Jeremy Wright,Ar	99%	NA
5		**Implementation**	**274 days**	**Mon 2/10/14**	**Thu 2/26/15**			**100%**	**Thu 2/26/15**
6		**Development Environment**	**270 days**	**Mon 2/10/14**	**Fri 2/20/15**			**100%**	**Fri 2/20/15**
7		Fabic Deployment	266 days	Fri 2/14/14	Fri 2/20/15		Jeremy Wright	100%	Fri 2/20/15
8		Django Project	0.5 days	Wed 2/19/14	Wed 2/19/14		Jeremy Wright	100%	Wed 2/19/14
9		ejabberd Configuration	29 days	Mon 2/10/14	Thu 3/20/14		Jeremy Wright	100%	Thu 3/20/14
10		Web Server Configuration	2 days	Wed 2/12/14	Thu 2/13/14		Jeremy Wright	100%	Thu 2/13/14
11		TastyPie Plugin Configuration	1 day	Mon 2/17/14	Mon 2/17/14		Jeremy Wright	100%	Mon 2/17/14
12		Python Virtual Environments	1 day	Tue 2/18/14	Tue 2/18/14		Jeremy Wright	100%	Tue 2/18/14
13		Testing	33.5 days	Wed 2/19/14	Mon 4/7/14	7,8,9,10,11,12	Jeremy Wright	100%	Mon 4/7/14
14		Development Documentation	31.5 days	Fri 2/21/14	Mon 4/7/14	13	Jeremy Wright	100%	Mon 4/7/14
15		**XMPP Implementation**	**31 days**	**Mon 2/24/14**	**Mon 4/7/14**	**6**		**100%**	**Mon 4/7/14**
16		Client Login/Auth	1 day	Fri 3/14/14	Fri 3/14/14	9	David Lucero	100%	Fri 3/14/14
17		**Tractor Demo Software**	**31 days**	**Mon 2/24/14**	**Mon 4/7/14**			**100%**	**Mon 4/7/14**
18		Command Language Grammar	3 days	Mon 2/24/14	Wed 2/26/14		David Lucero	100%	Wed 2/26/14
19		State Machine Design	3 days	Thu 2/27/14	Mon 3/3/14		David Lucero	100%	Mon 3/3/14
20		XMPP Client	8 days	Tue 3/4/14	Thu 3/13/14		David Lucero	100%	Thu 3/13/14
21		Testing	23 days	Thu 3/6/14	Mon 4/7/14	18,19,20	David Lucero	100%	Mon 4/7/14

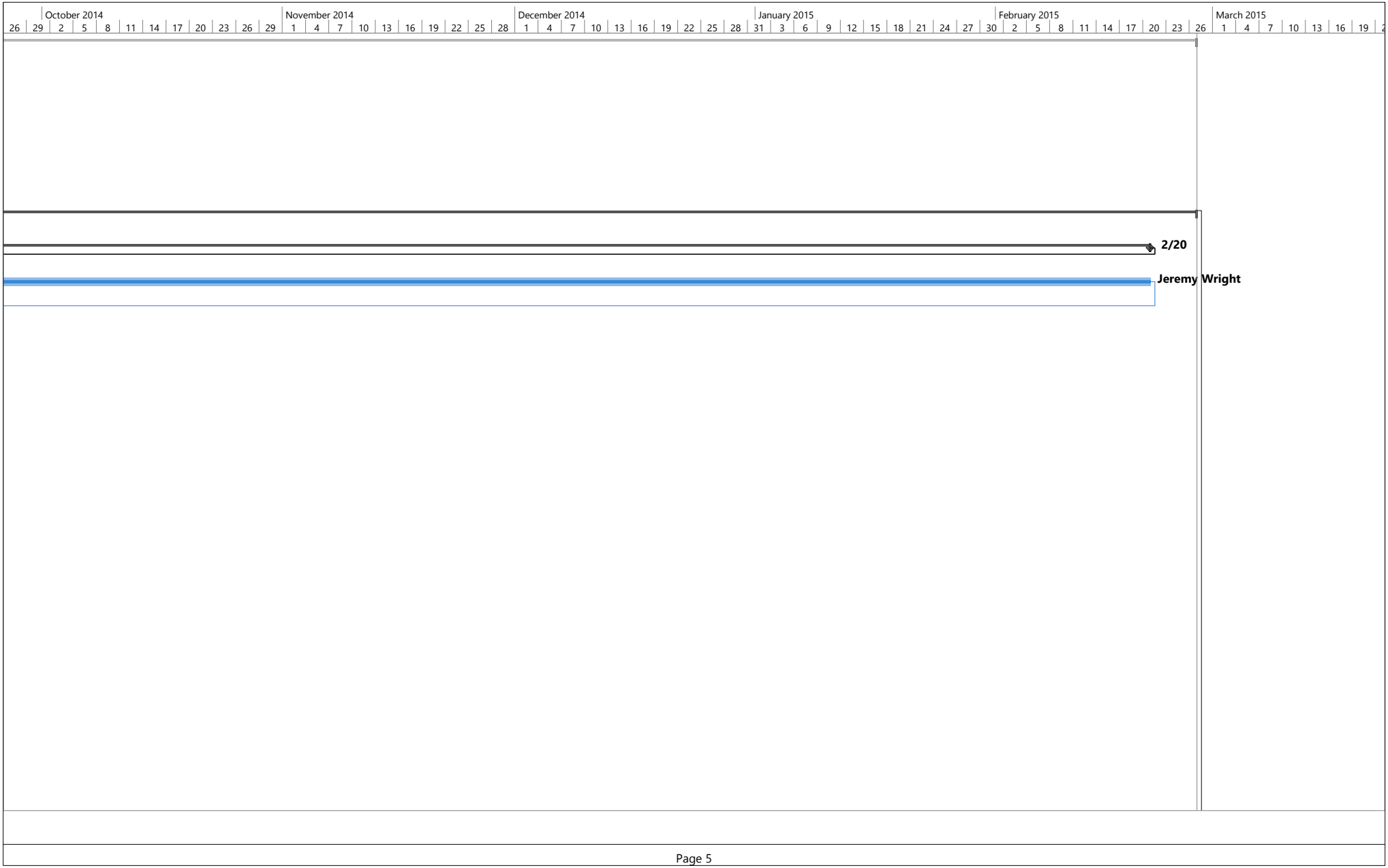


ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names	% Work Complete	Actual Finish
22		Debug Channel for Demo and Debug	14 days	Mon 3/17/14	Thu 4/3/14	17	Jeremy Wright	100%	Thu 4/3/14
23		Testing	14 days	Wed 3/19/14	Mon 4/7/14	22,16,17	Arun Balaji Budur	100%	Mon 4/7/14
24		**Frontend UI**	**41 days**	**Mon 2/10/14**	**Mon 4/7/14**			**100%**	**Mon 4/7/14**
25		Google Maps Rendering of Tractor Positions	1 wk	Mon 2/10/14	Fri 2/14/14		Arun Balaji Buduru	100%	Fri 2/14/14
26		Connect UI to REST Interface	9 days	Mon 3/10/14	Thu 3/20/14		Arun Balaji Buduru	100%	Thu 3/20/14
27		Mechanism to input a path to send to the Tractor	3.8 wks	Mon 2/24/14	Thu 3/20/14		Arun Balaji Buduru	100%	Thu 3/20/14
28		Testing	12 days	Fri 3/21/14	Mon 4/7/14	25,26,27	Arun Balaji Budur	100%	Mon 4/7/14
29		**Server Backend**	**19 days**	**Wed 3/12/14**	**Mon 4/7/14**	**6**		**100%**	**Mon 4/7/14**
30		XMPP to Django (Punjab) connection	3 days	Wed 3/12/14	Fri 3/14/14		Arun Balaji Buduru	100%	Fri 3/14/14
31		System Models	1 day	Mon 3/17/14	Mon 3/17/14		Arun Balaji Budur	100%	Mon 3/17/14
32		Testing	15 days	Tue 3/18/14	Mon 4/7/14	30,31	Arun Balaji Budur	100%	Mon 4/7/14
33		**REST Interface**	**264 days**	**Mon 2/24/14**	**Thu 2/26/15**	**6**		**100%**	**Thu 2/26/15**
34		API Design	264 days	Mon 2/24/14	Thu 2/26/15		David Lucero,Jere	100%	Thu 2/26/15
35		Authentication Model	250 days	Fri 3/14/14	Thu 2/26/15		David Lucero,Jeremy	100%	Thu 2/26/15
36		Connection to Punjab Models	3 days	Wed 3/19/14	Fri 3/21/14		David Lucero,Jeremy	100%	Fri 3/21/14
37		Testing	28 days	Thu 2/27/14	Mon 4/7/14		David Lucero,Jere	100%	Mon 4/7/14
38		System Testing	1.8 wks	Wed 3/26/14	Mon 4/7/14	5FF	Arun Balaji Budur	100%	Mon 4/7/14




April 2014
May 2014
June 2014
July 2014
August 2014
September 2014
uduru,David Lucero
Jeremy Wright,Arun Balaji Buduru,David Lucero
Jeremy Wright,Arun Balaji Buduru,David Lucero
t
Jeremy Wright
Jeremy Wright
4/7
David Lucero


pril 2014
May 2014
June 2014
July 2014
August 2014
September 2014
Jeremy Wright
Arun Balaji Buduru
4/7
duru
duru
Arun Balaji Buduru
Arun Balaji Buduru
,Jeremy Wright
David Lucero,Jeremy Wright
Arun Balaji Buduru,David Lucero,Jeremy Wright


October 2014
November 2014
December 2014
January 2015
February 2015
March 2015
26 29 2 5 8 11 14 17 20 23 26 29 1 4 7 10 13 16 19 22 25 28 1 4 7 10 13 16 19 22 25 28 31 3 6 9 12 15 18 21 24 27 30 2 5 8 11 14 17 20 23 26 1 4 7 10 13 16 19
2/20
Jeremy Wright


October 2014
November 2014
December 2014
January 2015
February 2015
March 2015
26 29 2 5 8 11 14 17 20 23 26 29 1 4 7 10 13 16 19 22 25 28 1 4 7 10 13 16 19 22 25 28 31 3 6 9 12 15 18 21 24 27 30 2 5 8 11 14 17 20 23 26 1 4 7 10 13 16 19
David Lucero,Jeremy Wright
David Lucero,Jeremy Wright