Robot Web Tools: Efficient Messaging for Cloud Robotics

Russell Toris[1], Julius Kammerl[2], David V. Lu[3], Jihoon Lee[4], Odest Chadwicke Jenkins[5], Sarah Osentoski[6], Mitchell Wills[1], and Sonia Chernova[1]

***Abstract*— Since its official introduction in 2012, the Robot Web Tools project has grown tremendously as an open-source community, enabling new levels of interoperability and portability across heterogeneous robot systems, devices, and front-end user interfaces. At the heart of Robot Web Tools is the *rosbridge* protocol as a general means for messaging ROS topics in a client-server paradigm suitable for wide area networks, and human-robot interaction at a global scale through modern web browsers. Building from *rosbridge*, this paper describes our efforts with Robot Web Tools to advance: 1) human-robot interaction through usable client and visualization libraries for more efficient development of front-end human-robot interfaces, and 2) cloud robotics through more efficient methods of transporting high-bandwidth topics (e.g., kinematic transforms, image streams, and point clouds). We further discuss the significant impact of Robot Web Tools through a diverse set of use cases that showcase the importance of a generic messaging protocol and front-end development systems for human-robot interaction.**

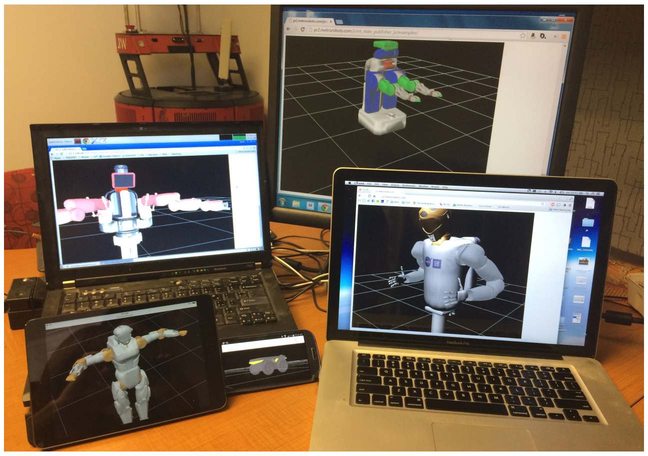


Fig. 1. The standard Robot Web Tools 3D user interface on multiple operating systems with multiple robots and cloud simulation instances. Clockwise from top right, PR2 on Windows, NASA Robonaut 2 on OS X, Clearpath Jackal on Android phone, NASA Valkyrie on iOS iPad, and Rethink Baxter on Ubuntu.

I. INTRODUCTION

The recent rise of robot middleware and systems software has drastically advanced the science and practice of robotics. Similar to a computing operating system, robot middleware manages the interface between robot hardware and software modules and provides common device drivers, data structures, visualization tools, peer-to-peer message-passing, and other resources. Advances in robot middleware have greatly improved the speed in which researchers and engineers can create new systems applications for robots by bringing a "plug-and-play" level of interoperability and code reuse. Projects in the space have supported a variety of cross-language and cross-platform tools that promote common functionality and best practices of component-based software design.

Among these efforts, the Robot Operating System (ROS) [1] has become the *de facto* standard middleware for enabling local area network control of robot systems. Developed initially for use with the Willow Garage PR2 robot, ROS is an open-source effort to provide the network protocols, build environment, and distribution release pipelines for researchers and developers to build robot systems and applications without re-writing standard functionality for every application or robot. Since its introduction, the number of individuals, laboratories, companies, and projects using ROS has grown tremendously [2]. This ROS community has also developed important general-use libraries for collision-free motion planning, task execution, 2D navigation, and visualization that have proved invaluable standards in robotics research.

Despite its many advantages, ROS and robot middleware in general has many shortcomings with regards to usability/accessibility, security, portability, and platform dependencies. For ROS, the only officially supported operating system is Ubuntu, a popular Linux distribution. Given that Ubuntu is used by less than two percent of worldwide computer users [3], ROS remains largely aimed for prototyping lower-level software systems. While there are projects that provide experimental support for operating systems other than Ubuntu, such as OS X and Arch Linux [4], those platforms are not fully stable and require specific support for the intricacies of each OS. As such, ROS in its current form is neither a standard nor sufficient for development end-user robot applications for human-robot interaction research.

In terms of cloud robotics, ROS largely assumes distributed computing over a local area network (LAN). Though

[1]Toris, Wills, and Chernova are with the Department of Computer Science at Worcester Polytechnic Institute {rctoris,mwills,soniac}@wpi.edu
[2]Kammerl was formerly a research scientist at Willow Garage at the time of this work julius@kammerl.de
[3]Lu is a PhD researcher in the Department of Computer Science at Washington University in St. Louis. davidlu@wustl.edu
[4]Lee is a Research Engineer at Yujin Robot Co.,Ltd jihoonl@yujinrobot.com
[5]Jenkins is faculty in the Department of Computer Science at Brown University cjenkins@cs.brown.edu
[6]Osentoski is a researcher at Robert Bosch LLC. sarah.osentoski@us.bosch.com

distributed in theory[1], the ROS middleware uses a centralized system for keeping track of peer-to-peer handshaking and globally relevant information like kinematic transforms, parameters, robot state, etc. Consequently, typical ROS operation proved problematic for cases that require remote communications over wide area and bandwidth limited networks.

Addressing these shortcomings and others, we present an analysis of the Robot Web Tools (RWT) project in terms of its ongoing uses and applications across robotics, present advancements for communication of robotics-oriented big data, and future directions addressing improved network efficiency. Initially proposed in [6], RWT is an ongoing effort to realize seamless, general, and implementation-independent applications layer network communications for robotics through the design of open-source network protocols and client libraries. A principal goal of RWT is to converge robot middleware (i.e. ROS) with modern web and network technologies to enable a broadly accessible environment for robot front-end development, cloud robotics, and human-interaction research suitable for use over public wide area networks. Instead of relying on the complex network configuration required by ROS and other cloud robotics paradigms, we have extended the generic *rosbridge* protocol [7] of RWT that can operate over a variety of network transports, including the web-accessible *WebSockets* and considerably improved the interoperability of robot systems, as demonstrated by the openEASE project (see Figure 1).

Our analysis of Robot Web Tools in this paper will cover: 1) the design of the RWT client-server architecture and use for developing web-accessible user interfaces for robots, 2) technical challenges and improved methods for network transport of high-bandwidth ROS topics, such as for articulated kinematics transforms, point clouds, and image streams, and 3) a survey of broader adoption and use of RWT in applications across robotics research and development.

II. RELATED WORK

Many robotics researchers have utilized Internet and Web technologies for remote teleoperation, time sharing of robot experimentation, data collection through crowdsourcing, and the study of new modes of human-robot interaction. As early as 1995, researchers enabled remote users to control a robotic arm to manipulate a set of colored blocks over the Web [8]. Goldberg enabled web users to maintain and monitor a garden, in which they could control a robot to perform tasks such as plant seeds and water plants [9]. Crick et al. presented remote users with a maze through which they were able to navigate using a robot, enabling the researchers to study how different levels of visual information affected successful navigation [10]. Building on these earlier efforts, a growing number of projects have recently emerged to allow researchers and end-users access to complex mobile manipulation systems via the web for remote experimentation [11], [12], human-robot interaction studies [13], and imitation learning [14].

[1]From the ROS documentation[5]: "there must be complete, bi-directional connectivity between all pairs of machines, on all ports."

More broadly, the field of *cloud robotics* research focuses on systems that rely on any form of data or computation from a network to support their operation [15]. Using this definition, a number of cloud robotics architectures have been proposed [16], [17], [18] aimed to offload as much computation as possible onto a "remote brain" unbounded by normal resource limits. One example is the RoboEarth project [18] which utilizes the Rapyuta cloud engine [19]. Rapyuta was developed as a way of running remote ROS processes in the cloud , using a *rosbridge*-inspired protocol of JSON-contained ROS topics.

While this definition creates a convenient vision of "cloud-enabled robots", it casts cloud robotics purely as a collection of proprietary back-end services without regard to broader interoperability and human accessibility. In contrast, RWT considers a more inclusive definition for "robot-enabled clouds" that also considers human-accessible endpoints and interfaces in a network, such as web browsers. With continued advances in robotics-oriented protocols, cloud robotics has the potential to further both human-robot interaction and expanded off-board computation in concert.

III. DESIGN AND ARCHITECTURE

In contrast to the distributed publish-subscribe architecture of ROS, RWT was designed to be a client-server architecture. The RWT client-server architecture emulates the design of the earlier Player/Stage robot middleware [20], [21] as well as the dominant paradigm for web development. Client-server systems separate a computational workload and resources into services. These services have providers (servers) and consumers (clients), where client processes request services from a server process. A wide variety of services can be provided such as to interface with a device, query for certain information, compute special routines or functions, and more, forming a cascade of services that ultimately lead to the user interface. Similar to the remote procedure call, client-server processes are commonly request-reply interactions communicated by message passing over networks. Broadcast and stream-oriented services, common to publish-subscribe in ROS, can also be supported in the client-server design.

The core element of client-server architectures is an established communications protocol that allows the implementation specifics of services to be abstracted. Such protocols enable clients and services to effectively speak the same language purely through messages. It is this message-based interoperability that has been the engine of growth for the Internet (via the IP protocol) and the World Wide Web (via the HTTP protocol). In contrast, ROS uses an informal protocol (TCPROS) with typed data messages (ROS topics) but no established definition. As such, ROS is more suited to being a build and distribution system for software instead of a general message passing architecture.

The established protocol for RWT is *rosbridge* [7], [22]. *rosbridge* communicates ROS data messages contained in the JavaScript Object Notation (JSON) for straightforward marshalling and demarshalling. Below is an example *rosbridge*

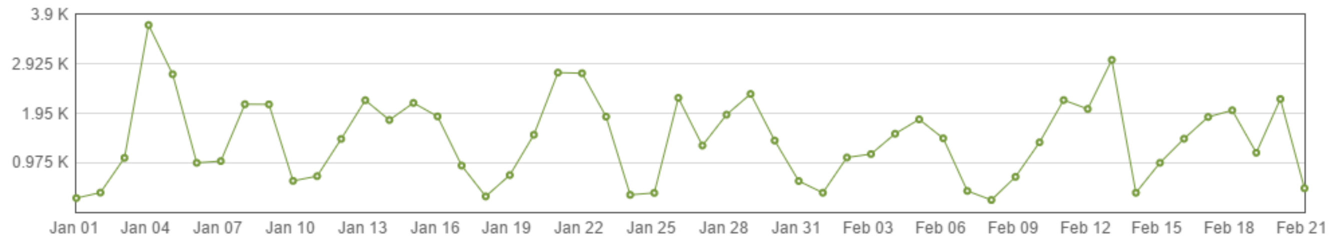


Fig. 2. Number of HTTP Requests to the CDN Libraries for 2015

message for a 6-DOF velocity command for a robot to move forward encoded in JSON:

```
{ "op": "publish",
  "topic": "/cmd_vel",
  "msg": {"linear":{"x":1.0,"y":0,"z":0},
           "angular":{"x":0,"y":0,"z":0}
}
```

Through its use of WebSockets (a protocol built on top of HTTP), *rosbridge* can be readily used with modern web browsers without the need for installation. This fact combined with its portability and pervasive use makes the web browser an ideal platform for human-robot interaction. RWT 1.0 included a simple lightweight client library known as `ros.js` [22] and a user interface and visualization package known as WViz (web-visualizer) [12], [11]. `ros.js` provided basic ROS functionality to browser-based programs via the early version of the *rosbridge* protocol, known now as *rosbridge* v1.

In RWT 2.0, the *rosbridge* v2 protocol[2] added the ability to communicate more efficiently (e.g., specification of compression types) and provide near-complete access to ROS functionality. RWT 2.0 allowed for more elaborate, responsive, and reliable interfaces to be built [6] and stricter security measures regarding access by unauthorized remote clients [23].

To address a wide range of use cases and applications development for HRI, the three JavaScript libraries were developed to facilitate web-based human-robot interaction: the *roslibjs* client library and *ros2djs* and *ros3djs* visualization libraries. The design of these libraries enables easy and flexible development of robot user interfaces that are fast and responsive. These libraries avoid both a large monolithic structure (which can be bandwidth intensive like WViz) and a large package system (which breaks out functionality into too many separate libraries like ROS). These cases were avoided principally due to the design of *roslibjs* to be used entirely without any visualization dependencies. Standalone widgets and tools, such as for navigation and mobile manipulation, can be built on top of these libraries and distributed as such, as described below:

roslibjs is the client library for communication of ROS topics, services, and actions. *roslibjs* includes utilities for common ROS and robotics tasks such as transform (TF) clients, URDF (Unified Robot Description Format) parsers, and common matrix/vector operations, as well as fully compatibility with server-side JavaScript environments (such as *node.js*).

ros2djs and *ros3djs* are the visualization libraries for ROS-related 2D and 3D data types, such as robot models, maps, laser scans, and point clouds, and interaction modes, such as interactive markers [24]. Both libraries build on existing libraries for 2D (*EaselJS*) and 3D (*three.js*) rendering on the HTML5 `<canvas>` element.

To make releases of the libraries easy and efficient to use, pre-built and compressed JavaScript files are uploaded to a public CDN (content delivery network). Such a system makes use of multiple cache servers around the globe to serve clients the libraries as efficiently as possible. These servers manage thousands of hits a week without putting a load on a single failure point. Figure 2 showcases typical hits for the hosted RWT libraries for 2015.

IV. EFFICIENT MESSAGING OF HIGH-BANDWIDTH TOPICS

Although RWT is designed for ease of comprehension and use, the challenges for creating a robust and reliable systems has many open questions. Most prominent is the challenge of real-time control. While new emergent technologies, such as WebRTC (real-time communication), are helping to improve efficient communication to and from standard web browsers [25], these technologies are still experimental, in constant flux, and widely unsupported at the time of this writing (these ideas are further discussed in Section VI).

The main problem stems from the amount of data typically associated with streaming robot sensors and visualization information in order to minimize latency and permitting usable supervisory control. Support for "real-time" communication (e.g., for live streaming) in modern web browsers trade-off latency due to buffering for smoother streams. Such buffers add an extra delay (typically in the magnitude of seconds) which cause the robot interface to seem non-responsive and confusing to operators.

Further, due to security and other design constraints, it is not possible to create raw TCP or UDP sockets from JavaScript in a browser. As such, the most common communication approach for bi-directional client-server communication is with a WebSocket connection. WebSockets are built on-top of HTTP and leave an open communication

[2]`https://github.com/RobotWebTools/rosbridge_suite/blob/develop/ROSBRIDGE_PROTOCOL.md`

channel to a WebSocket server. As a consequence to this, communication is limited to the efficiency and robustness of HTTP and, thus, unable to send raw or binary data.

Given these constraints, efficient communication of robot data is critical. We discuss advances made by RWT to overcome these limitations for several high-bandwidth message types for transform subscriptions, image streaming, and point cloud streaming, as well as message compression.

A. *Transform Subscriptions*

In ROS, information about the multitude reference frames in robot environments is communicated via a stream of transform data, specifying how to translate one frame into another, creating a large, constantly updated tree of geometric relations. This behavior is regulated by a library known as *tf* and *tf2* [26]. Since many of the transforms are likely to be constantly changing and operating on stale data could lead to problematic behaviors, the *tf* library requires that all relevant transforms be constantly published. Any component of the system can publish additional transforms, making the system neatly decentralized.

While the constant stream of data creates a robust environment for many applications, it is not a low-bandwidth-friendly solution. The sheer amount of data published can be overwhelming. As Foote notes, the transforms for the PR2 robot can take up to 3Mbps of bandwidth [26]. Furthermore, much of this data is redundant. For example, if two components of a robot are fixed in relation to one another, the transform between them will still need to be published at the same rate as the rest of the TF tree in order to keep the entire tree up to date.

To reduce the amount of bandwidth devoted to *tf*, RWT employs the package *tf2_web_republisher*. This is a ROS node that uses ROS' action interface to pre-compute requested transforms on demand, which can then be published via *rosbridge* with a smaller bandwidth usage. Transforms are published in reference to some specified global frame at a reduced rate and only when a link is changed within some delta. The inclusion of this node simply moves computations that usually happen within a specific node to happen within a central authority. The result is the same effective information being provided to remote clients, without the same bandwidth usage.

To illustrate this point, bandwidth usage of TF data over a one minute period was captured using both the traditional ROS TF subscription and the web-based subscription. While the PR2 robot mentioned earlier has been shown to utilize 3Mbps of bandwidth, it can be argued this is due to the complexity of number of DOFs the PR2 contains. Here, a more simplified mobile robot was used with a 6-DOF arm which contained 52 frames. During the capture, the robot's base and arm were teleoperated to trigger TF republishes. For this capture, the web republisher was set to a default rate of 10.0Hz with change delta values of 0.01 meters and 0.01 radians. Figure 3 visualizes the results from the capture. Note that the ROS TF rate had an average of 208.5 KB/s ($\sigma = 1.5$) while the web based TF rate was had an average of 96.0 KB/s ($\sigma = 40.6$).

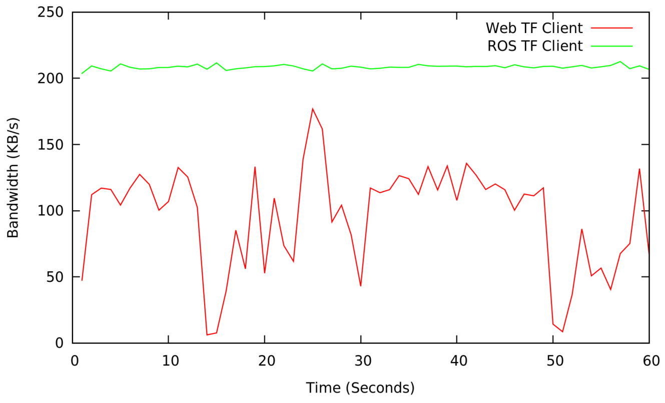


Fig. 3. Average TF Bandwidth from a ROS and Web Based Client

B. *Image Streams*

Another bandwidth-heavy data stream is any sort of video. In ROS, three main image streaming techniques are used as part of the image stream pipeline: raw images which utilize a ROS image message, compressed images which utilize JPEG or PNG image compression, and Theora streams which utilize the Theora video codec. As mentioned earlier, transport types to the browser are limited. While it would be possible in theory to send raw ROS images over *rosbridge*, the overhead and bandwidth of transporting images in their text JSON representation would be impracticable.

To overcome this limitation, RWT introduced two transport options which are available to all, or a large subset of browsers[3]. The first option utilizes MJPEG streams which can be embedded in any HTML `<img>` tag. ROS images are transformed into series of JPEG images and streamed directly to the browser. While more efficient options exist, this transport type is available to all modern web browsers. The second streaming option utilizes the newer HTML5 `<video>` tag. This tag allows for more efficient video codecs to be streamed to a browser. In particular, the VP8 codec standard is used within WebM streams. The main limitation here is its limited support on modern browsers (typically well supported on Chrome and Firefox). Furthermore, buffering within the video codecs results in an unavoidable delay or lag in transmission.

To illustrate bandwidth requirements from ROS to a browser, we provide an average bandwidth for multiple transport types across a 2 minute capture using a 640x480 pixel image at 30Hz from a USB camera. MJPEG compression is set to the default value of 90% while VP8 was set to a default bitrate of 100,000. Results are presented in Table I.

C. *Point Cloud Streams*

One of the most bandwidth intensive data types in ROS, robotics, and computer vision are 3D point clouds. This depth

[3]Transports are available via the *web_video_server* ROS package.

TABLE I

Bandwidth Usage of Image Streams via Various Transport Types in KB/s

	Web		ROS Internal		
	MJPEG	VP8	RAW	Compressed	Theora
μ	1317.0	342.2	12052.5	1309.3	19.4
σ	11.1	18.3	1.7	11.6	8.9

information, typically overlayed with corresponding RGB data to form RGBD data, contains much more information than a video or image stream making it extremely useful for both computational algorithms as well as visualization. Standard compression types for point clouds are limited and certainly not available to web browsers. As such, new techniques are needed to allow point clouds to be streamed to the browser efficiently for visualization purposes.

As with image streams, while it would be possible to stream raw point cloud data across *rosbridge* in a JSON encoded message, the overhead and bandwidth requirements would be too costly to use effectively, even with message compression. In order to make full use of the built in compression and streaming features available in modern web browsers, we have developed a method for encoding RGBD depth maps as a single image stream which can be sent across the wire using HTML5 video codecs and extracted in the browser for visualization.

The key to this idea is that point clouds are not just used for computation, but also visualization by people. As such, lossy compression types can be used to dramatically reduce bandwidth while still maintaining a visually useful display. This compression is done by streaming a single image composed of three separate "images" (shown in Figure 4(a)): encoded depth information from 0 to 3 meters, encoded depth information from 3 to 6 meters, a binary mask indicating valid sample points and the RBG image. Encoded depth information is split into two frames (0-3 and 3-6) in order increase the range of depth information that can be stored in an image (i.e., a finer discretization). To increase compression rates, compression artifacts are reduced by filling areas of unknown depth (typically represented by `NaN`) with interpolated sample data. On the client side, the embedded binary mask can then used to omit these samples during decoding. This single image is streamed to the web browser over a VP8 codec (via the *web_video_server* mentioned previously) and is assigned to a WebGL texture object with a vertex shader which allows for fast rendering of the point cloud via the clients GPU. An example of the resulting compressed point cloud image is shown in Figure 4(b).

To illustrate bandwidth requirements, we provide an average bandwidth for multiple transport types across a 2 minute capture using an ASUS Xtion Pro RGBD camera. We again use the default compression values stated earlier. Results are presented in Table II.

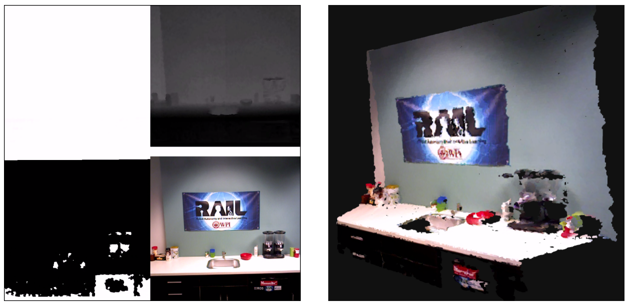

(a) The Encoded Point Cloud Image (b) The Resulting Rendering

Fig. 4. Point Cloud Streaming

TABLE II

Bandwidth Usage of Point Cloud Streaming in KB/s

	ROS Internal	Web Streaming
μ	5591.6	568.4
σ	70.5	133.5

D. Generic Message Compression

Besides transform, image, and point cloud data, there exists many other formats of high bandwidth data in ROS. High resolution map data used for robot navigation is often larger than 2MB per message. 3D visualization and interactive markers [24] often contain large sets of points needed to render 3D models in a visualizer. Furthermore, since user-defined data structures are supported in ROS, a potentially unbounded amount of data could be sent through a ROS message.

To support large data streams in the general manner, RWT allows for compression options to be set. In particular, we utilize the web browsers built in PNG de-compression to efficiently package and send large messages in a lossless manner. If a client specifies that a data stream be sent using PNG compression, the *rosbridge* server will start by taking the raw bytes of the JSON data (in its ASCII representation) and treat them as the bytes of an RGB image. Padding is added with null values to create a square image. This image data is then compressed using standard PNG compression libraries and sent across the wire using base 64 encoding. This data can then be read internally by the browser and used to extract the original RBG values which, in tern, represent the ASCII JSON encoding of the message itself.

To illustrate the effect this has on message compression, we transmit a series of typical large format ROS messages using PNG compression. Results are presented in Table III which show the original size in KBs, the compressed size in KBs, and the time to compress and transmit in milliseconds. As a result, it reduces data less than 30% of its original size with negligible time loss.

V. SURVEY OF USE CASES

Since its original introduction in 2012 [6], RWT has been successfully deployed across multiple robot platforms and a wide range of applications. We posit such significant

TABLE III

PERFORMANCE OF PNG COMPRESSION FOR ROS MESSAGES

	Original	Compressed	Time
Map (732x413)	2725	39	40.0
3D Point Array Marker	42.1	5.1	0.8
3D Line Strip Marker	42.1	4.8	0.9
3D Line List Marker	78.1	7.7	1.0

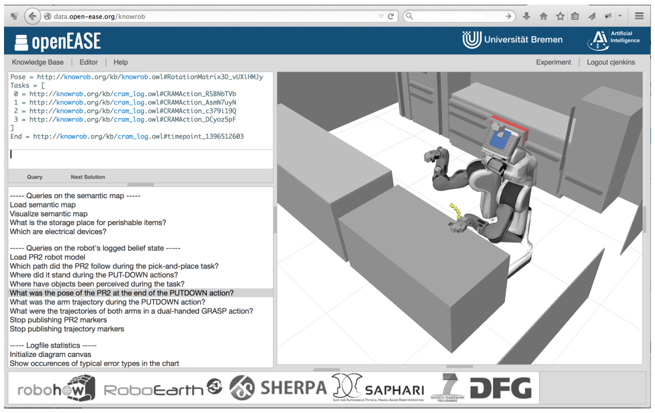
use demonstrates the need for and impact of open-source network protocols for robotics, human-robot interaction, and cloud robotics. The following is a small sampling of projects the RWT open-source releases[4] in compelling research and development projects.

One of the earliest applications was for the Robots for Humanity project, which seeks to develop robotic technologies to help people overcome physical disabilities [27]. For such applications, users often require custom interfaces that work on a wide variety of plaforms and several design iterations to work around physical constraints, which can more readily be done through RWT than other alternatives. In this project, a person with a mute quadriplegic disability collaborated with a team of researchers to develop interfaces that utilized both the native ROS visualizer and RWT to enable accessible control of a PR2 and a quad rotor to people with disabilities. Through these tools, this user was able operate a robot over the web to shave himself, fetch a towel, open a refrigerator to retrieve yogurt, and inspect his yard. Another independent group of researchers has also applied RWT to the problem of developing a web-based robot control inteface for quadriplegics [28]. Utilizing the Baxter robot, this project explores the design of single-switch scanning interfaces for controlling the robot end effector.

RWT has also been applied to general purpose mobile manipulation tasks, with both simulated and real-world robots. Ratner et al. have developed a web-based interface that is capable of quickly recording large numbers of high-dimensional demonstrations of mobile manipulation tasks from non-experts [29]. Coupled with a light weight simulated environment, the tool is capable of supporting 10 concurrent demonstrators on a single server, leveraging existing crowd-sourcing platforms to perform large scale data collection. Knowledge obtained in simulation is then transferred onto the physical robot, such as from a simulated to a real-world PR2.

Similarly, the OpenEASE project [30] also makes use of simulation and web-based interface to provide a remote knowledge representation and processing service that aims at facilitating the use of Artificial Intelligence technology for equipping robots with knowledge and reasoning capabilities. Unlike [29], which seeks to advance the mobile manipulation capabilities of a single robotic system from demonstration data, OpenEASE analyzes experiences of manipulation episodes and reasons about what the robot saw, what it did, how it did that, why, and what effects it caused. This

[4]https://github.com/RobotWebTools



```
Pose = http://knowrob.org/kb/knowrob.owl#RotationMatrix3D_vUXiHMJy
Tasks = [
 0 = http://knowrob.org/kb/cram_log.owl#CRAMAction_RSBNbTVb
 1 = http://knowrob.org/kb/cram_log.owl#CRAMAction_AsmN7uyN
 2 = http://knowrob.org/kb/cram_log.owl#CRAMAction_c379i19Q
 3 = http://knowrob.org/kb/cram_log.owl#CRAMAction_DCyoz5pF
]
End = http://knowrob.org/kb/cram_log.owl#timepoint_1396512603
```

----- Queries on the semantic map -----
Load semantic map
Visualize semantic map
What is the storage place for perishable items?
Which are electrical devices?

----- Queries on the robot's logged belief state -----
Load PR2 robot model
Which path did the PR2 follow during the pick-and-place task?
Where did it stand during the PUT-DOWN actions?
Where have objects been perceived during the task?
What was the pose of the PR2 at the end of the PUTDOWN action?
What was the arm trajectory during the PUTDOWN action?
What were the trajectories of both arms in a dual-handed GRASP action?
Stop publishing PR2 markers
Stop publishing trajectory markers

----- Logfile statistics -----
Initialize diagram canvas
Show occurences of typical error types in the chart

Fig. 5. The openEASE project by Tenorth, Beetz, et al. is an example of robot operation and semantic mapping through RWT.

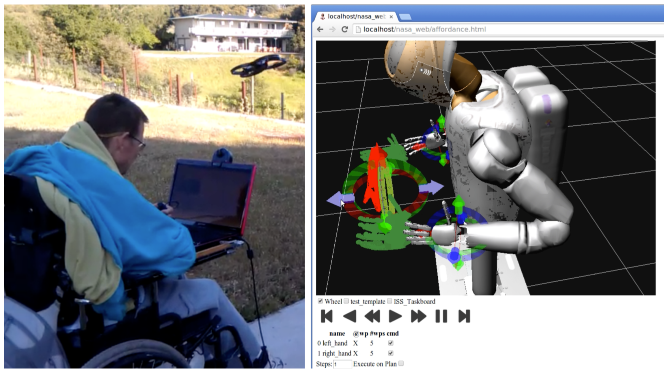


Fig. 6. Browser-based control of a (left) quad rotor and (right) a simulation of the NASA Robonaut 2.

information is then stored in a platform-independent way, and can then be retrieved by queries formulated in PROLOG, a general-purpose logic programming language. The system can be used both by humans via a web-based graphical interface, or by robots that use OpenEASE as a cloud-based knowledge base via a webservice API.

Additional use cases have been found in commercial applications. Savioke, a startup creating a robot hotel butler, uses RWT to bridge between ROS, used to control/monitor the robot, and a web layer, which implements the application logic (e.g. the delivery application) as well as the user interfaces. Rethink Robotics runs *rosbridge* automatically, which enables users to interface with the robot without needing a ROS installed platform.

Furthermore, while JavaScript and the web browser are the chief clients used in conjunction with *rosbridge*, the protocol *rosbridge* provides also creates opportunities for communicating with other platforms that do not have robust ROS implementations of their own. For example, the *jrosbridge* standalone library provides communication with the ROS platform in native Java allowing for ROS communication on a number of different operating systems and architectures. There are also separate projects for using the *rosbridge* protocol to communicate with MATLAB and LabView code. It also provides a low-footprint communication platform for

Arudino projects.

VI. FUTURE TECHNOLOGIES

With the continued development of new web standards, the capabilities of web applications continue to grow. New standards allow for web applications to interact more with the hardware of the device that it is running on, create more complex interfaces, and communicate using new robust and efficient protocols. Utilizing these standards can allow for building ROS web applications with advanced features, while still offering the platform independence and ease of use that comes from the web. Additionally, the continued development of these standards reduces the need for plug-ins that users would previously have needed to install (such as Adobe Flash or Microsoft Silverlight) in order to obtain more access to the underlying computer or perform more complex computational and network operations.

APIs that allow for access to the sensors and input devices on a device allows for interfaces to be built that feel more like a native application. Through the use of new sensor APIs a web application can now have access to the device acceleration and orientation. These sensors are becoming available as the number of portable electronics (smart phones, tablets, etc.) increases. This can allow for interfaces where the user can move their device around to interact, instead of only clicking or touching on the screen. Other APIs also exist to get the state of a device such as battery status, geolocation, and improved touch screen support. These new APIs also open up access to audio and video devices from the web browser. A web application can now stream audio and video from the user's computer's webcam and microphone in addition to streaming video to the browser.

In addition to simply allowing for more information to be accessed from the browser, new APIs also allow for an application to do things more efficiently and concisely. Some of the APIs that is already used by the RWT client libraries are the canvas and WebGL APIs which allow for high performance 2D and 3D graphics in the browser. Web Workers are another API that allows for running tasks in the background. JavaScript usually runs in a single thread which means that performing a complex operation in JavaScript can cause the UI to lag. Currently if roslibjs receives a large number of messages or very large messages it must serialize/deserialize them in the main JavaScript thread, which can block the UI. If Web Workers were used then these operation could be performed in the background and potentially even in parallel allowing for a more responsive UI. Additionally, because web applications can do things in the background it means that they do not have to be a thin client any more and can do processor intensive work.

Furthermore, new web technologies are not just allowing a better web application, but they also allow for better communication with a robot. Initial web technologies were not built for the high bandwidth low latency applications that are being developed today. In order to build an immersive interface for the user, a lot of information (real-time video, point clouds, etc) must be streamed to the user with a minimal delay. As discussed earlier, current solutions for streaming this kind of data in RWT involve streaming the information as a compressed video (MJPEG or VP8) to the client over HTTP. However, because TCP (which HTTP is built on) is a reliable transport protocol, all of the information must be successfully transferred to the client which is not always necessary for responsive visualization purposes. New web standards, such as WebRTC[25], are looking to enable developers to build more media intensive applications. WebRTC allows for the transport of video over UDP and provides automatic adjustment of the video bitrate in response to changing available bandwidth. WebRTC not only allows for the transmission of real time video, but also supports audio, which opens up the possibilities for more feedback to the robot operator. In order to create more interactive applications WebRTC can also support streaming video and audio from the web browser, which could allow for the creation of a telepresence robot controlled purely from the browser.

A critical advance in WebRTC is that it can use direct peer to peer communication even if one of the peers is behind a router. This means that the robot could be hidden behind a NAT device with a server hosting the web application; that is, the robot's computer does not have to be directly exposed to the Internet. The standard not only supports the delivery of media, but also introduces a high-performance data channel API which is specifically designed for low-latency real-time communication. These channels can be either reliable or unreliable and implement congestion and flow control as well as encryption. The WebRTC standard has already been mostly implemented in Chrome, Firefox and Opera and support is planned for more browsers. It is also not just limited to desktop browsers, but is also supported by the mobile versions as well, which allows for the same technologies to be used on any form factor device. Current efforts are under-way to bring such standards into the core RWT libraries and tools allowing for responsive, immersive, and power portable robot web applications.

VII. CONCLUSION

This paper presents and overview, analysis, and discussion on how the Robot Web Tools project has enabled advances in cloud robotics and human-robot interaction research through the use of efficient and easy to use protocols. By overcoming challenges of high-bandwidth data flow for robot applications to the web, researchers are able to better develop and deploy applications and interaction modes to a wider audience on a wider range of platforms and devices as shown in a brief survey of known use cases.

One of the greatest benefits of open source software is the freedom to extend the code to do things that it was never originally planned to do. The work in this paper has enabled an unprecedented level of freedom for visualizing and interacting with ROS-based systems on a wide variety of new platforms. Running Robot Web Tools through the browser enables easy and quick interfaces on new platforms without the steep learning curve required for writing native ROS clients for each individual operating system.

It should not be understated how valuable having the interface design tool be a language that is as pervasive as JavaScript. With interactive websites growing in prevalence over the past decade, the number of interface developers using JavaScript, HTML5 and the related tools has grown as well. Coupled with the growing capabilities of the modern web browser, web interfaces have become a new standard.

RWT is a continuing and ongoing effort, like ROS and many open-source projects in robotics. We have discussed a number of applications for RWT, many of which were not initially imagined when designing the *rosbridge* protocol or the RWT client libraries. Possible future directions and web technologies were also described that can help RWT improve both cloud robotics and HRI. We encourage further extension and contributions to RWT to help broaden the accessibility and improve the interoperability of robotics for new web and network-based applications.

ACKNOWLEDGMENT

This work was funded in part by the National Science Foundation with grants IIS-1149876, IIS-1208497, and IIS-0844486, the Office of Naval Research with grant N00014-08-1-0910, NASA with grant NNX13AN07A, and the Ministry of Trade, Industry, & Energy (MOTIE, KOREA) Robot Industry Original Technology Development Project NO.10041789.

The authors also acknowledge and thank the various companies and individuals who helped to support and develop portions of the project including Brandon Alexander, David Gossow, Kaijen Hsiao, Alexander Sorokin, Nils Berg, Ben Pitzer, Trevor Jay, Jonathan Mace, Willow Garage Inc., and Robert Bosch LLC.

REFERENCES

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, Kobe, Japan, 2009.

[2] T. Foote, "ROS Community Metrics," 2014, http://download.ros.org/downloads/metrics/metrics-report-2014-07.pdf.

[3] "Operating System Market Share," http://marketshare.hitslink.com/operating-system-market-share.aspx.

[4] "ROS Hydro Installation Page," http://wiki.ros.org/hydro/Installation.

[5] "ROS Network Setup," http://wiki.ros.org/ROS/NetworkSetup.

[6] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris, "Robot Web Tools [ROS topics]," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 20–23, 2012.

[7] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: ROS for non-ros users," in *Proceedings of the 15th international symposium on robotics research (ISRR)*, 2011.

[8] K. Taylor and J. Trevelyan, "A Telerobot On The World Wide Web," in *1995 National Conference of the Australian Robot Association*. Melbourne: Australian Robotics Association, July 1995.

[9] K. Goldberg, Ed., *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*. Cambridge, MA, USA: MIT Press, 2001.

[10] C. Crick, S. Osentoski, G. Jay, and O. C. Jenkins, "Human and robot perception in large-scale learning from demonstration," in *Proceedings of the 6th ACM/IEEE International Conference on Human-Robot Interaction*, 2011.

[11] S. Osentoski, B. Pitzer, C. Crick, G. Jay, S. Dong, D. H. Grollman, H. B. Suay, and O. C. Jenkins, "Remote robotic laboratories for learning from demonstration - enabling user interaction and shared experimentation." *International Journal of Social Robotics*, vol. 4, no. 4, pp. 449–461, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijsr/ijsr4.html#OsentoskiPCJDGSJ12

[12] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. Jenkins, "Pr2 remote lab: An environment for remote development and experimentation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 3200–3205.

[13] R. Toris, D. Kent, and S. Chernova, "The robot management system: A framework for conducting human-robot interaction studies through crowdsourcing," *Journal of Human-Robot Interaction*, vol. 3, no. 2, pp. 25–49, 2014.

[14] M. J.-Y. Chung, M. Forbes, M. Cakmak, and R. P. Rao, "Accelerating imitation learning through crowdsourcing," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4777–4784.

[15] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *Automation Science and Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–12, 2015.

[16] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *Network, IEEE*, vol. 26, no. 3, pp. 21–28, 2012.

[17] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud networked robotics," *Network, IEEE*, vol. 26, no. 3, pp. 28–34, 2012.

[18] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. Van De Molengraft, "Roboearth," *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, June 2011.

[19] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 438–444.

[20] R. T. Vaughan, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2003, pp. 2421–2427.

[21] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[22] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. C. Jenkins, "Robots as web services: Reproducible experimentation and application development using rosjs," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 6078–6083.

[23] R. Toris, C. Shue, and S. Chernova, "Message authentication codes for secure remote non-native client connections to ROS enabled robots," in *Technologies for Practical Robot Applications (TePRA), 2014 IEEE International Conference on*, April 2014, pp. 1–6.

[24] D. Gossow, A. Leeper, D. Hershberger, and M. T. Ciocarlie, "Interactive markers: 3-d user interfaces for ros applications [ROS topics]," *Robotics Automation Magazine, IEEE*, vol. 18, no. 4, pp. 14–15, 2011.

[25] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "WebRTC 1.0: Real-time communication between browsers," *Working Draft, W3C*, vol. 91, 2012.

[26] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[27] T. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C.-H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama, "Robots for humanity: A case study in assistive mobile manipulation," *IEEE Robotics & Automation Magazine, Special issue on Assistive Robotics*, vol. 20, 2013. [Online]. Available: http://lifesciences.ieee.org/images/pdf/06476704.pdf

[28] V. Kalokyri, R. Shome, I. Yochelson, and K. E. Bekris, "A single-switch scanning interface for robot control by quadriplegics," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop on Assistive Robotics for Individuals with Disabilities: HRI Issues and Beyond*, 2014.

[29] E. Ratner, B. Cohen, M. Phillips, and M. Likhachev, "A web-based infrastructure for recording user demonstrations of mobile manipulation tasks," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[30] M. Beetz, M. Tenorth, and J. Winkler, “Open-ease a knowledge processing service for robots and robotics/ai researchers,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015, p. in press.