Real-Time Groupware in the Browser: Testing the Performance of Web-Based Networking

Carl Gutwin and Michael Lippold
Computer Science, University of Saskatchewan
110 Science Place, Saskatoon, SK, S7N 5C9
carl.gutwin, mike.lippold @usask.ca

T. C. Nicholas Graham
School of Computing, Queen's University
Kingston, ON, K7L 3N6
graham@cs.queensu.ca

ABSTRACT
Standard web browsers are becoming a common platform for delivering groupware applications, but until recently, the only way to support real-time collaboration was with browser plug-ins. New networking approaches have recently been introduced – based on re-purposed techniques for delivering web pages (Comet), or integration of real-time communication directly into the browser (HTML5 WebSockets). Little is currently known, however, about whether these new approaches can support real-time groupware. We carried out a study to assess the performance of the three different networking approaches, based on a framework of groupware requirements, in several network settings. We found that web-based networking performs well – better than plug-in approaches in some cases – and can support the communication requirements of many types of real-time groupware. We also developed two groupware applications using Comet and WebSockets, and showed that they provided fast and consistent performance on the real-world Internet. Our studies show that web-based networking can support real-time collaboration, and suggest that groupware developers should consider the browser as a legitimate vehicle for real-time multi-user systems.

Author Keywords
WWW, real-time groupware, performance, plug-ins.

ACM Classification Keywords
H.5.3 [Information Interfaces and Presentation]: *CSCW*.

General Terms
Design, Human Factors, Performance.

INTRODUCTION
The standard web browser is increasingly becoming a platform for delivering rich interactive applications, and many of these web-based applications are groupware: from office systems (e.g., Google Docs, docs.google.com) to instant messaging (e.g., Meebo, meebo.com), to multiplayer games (e.g., Stick Arena, xgenstudios.com). There is, however, a fundamental division in how these online applications are deployed: some are based on plug-in technologies such as Java (java.com), Flash (adobe.com/flash) or Silverlight (silverlight.net), and others are based on standards-based 'plain browser' technologies such as DHTML, AJAX, or Comet.

Of these techniques, only plug-ins have traditionally been used for synchronous groupware (i.e., systems in which people can see each other move and interact in the shared environment, in real time and at a high frame rate). The 'plain browser' approach is primarily used for semi-synchronous and asynchronous groupware (although some recent systems are moving towards real-time updates [8]).

One reason why synchronous groupware has only been attempted with plug-ins is that real-time interaction has much stricter network requirements (in terms of update rate, message throughput, and latency) than semi-synchronous applications, and web technologies have not traditionally been able to provide this level of performance. Recent advances in web-based networking, however, open the door to supporting real-time interaction in the plain browser. For example, capabilities in Ajax allow for streaming data to be sent from the web server to the browser, and a 'web socket' capability is part of the specification for HTML5 [26]. A groupware solution that does not require plug-ins is an important advance, because although plug-ins can work well, they present several disadvantages. First, the required plug-ins may not be installed, and it may not be possible for the user to install them (e.g., public-access machines or managed systems), whereas a browser is part of the standard software on most computers. Second, browsers on some devices do not support plug-ins at all (e.g., on the Apple iPhone and iPad); in these situations, the browser is the only option for web-based groupware. Third, many developers are advocating a move towards browser-only applications, to reduce dependence on third-party or proprietary plug-in providers [12]. For these reasons, more and more groupware applications will eventually be deployed on the standards-based browser.

Although web-based networking offers new opportunities, there is still relatively little information available about building synchronous groupware with techniques based on JavaScript, HTML5, AJAX, and Comet. There are several questions that need to be answered – and a primary one is

CSCW 2011, March 19–23, 2011, Hangzhou, China.

whether these mechanisms can meet the network requirements of synchronous groupware, in terms of update rate, message size, and latency.

In this paper we address this question by comparing the network performance of three different web-based networking approaches. The first approach re-purposes existing networking mechanisms that were originally designed for delivering web pages (i.e., Comet); the second integrates traditional networking mechanisms into the browser itself (i.e., WebSockets); and the third inserts a separate execution environment into the browser (i.e., the plug-in approach). To examine these three general approaches, we carried out network tests with AJAX polling, XHR multipart streaming, XHR iframe streaming, WebSockets, and Java applets. We also built two example groupware systems (a multi-user puzzle and a shared whiteboard) as a further test of web-based networking.

Our results show that 'plain browser' web networking can successfully support synchronous groupware. Comet techniques were able to sustain update rates of at least 20 messages per second in both LAN and WAN tests, and WebSockets were able to maintain much higher rates, even with large message sizes. Round-trip latencies for Comet techniques were between 67ms (LAN) and 185ms (WAN), and for WebSockets were between 11ms and 86ms. These performance rates are enough to support many common types of groupware, including shared editors, awareness systems, and multiplayer games. Our results suggest that web-based networking will not be the main limitation in delivering groupware in the browser – issues such as the speed of the real-world Internet or the performance of the browser's graphics subsystem are much more likely to be the limiting factor than the browser's network performance.

The two main contributions of this work are to establish that real-time interaction can be supported using standards-based web networking, and to specify the performance characteristics of three different approaches to supporting real-time work in the browser. Our results provide initial guidelines about what networking mechanisms to use when designing real-time groupware for deployment on the World-Wide-Web.

BACKGROUND

Our work builds on previous results from several communities including CSCW, WWW, and distributed systems. In CSCW, researchers have considered several problems in real-world development and deployment of groupware, and have presented toolkits (e.g., [2,5,17,25]), techniques for improving robustness (e.g., [3,11]), and some performance tests (e.g., [7,10]). In addition, several groupware systems have been built on the Web, but these are primarily asynchronous collaboration tools (e.g., [1]). A few recent examples show real-time groupware systems built with Comet techniques [14,22], but few performance evaluations have been conducted.

Researchers in games and distributed systems have carried out extensive work in testing many aspects of real-world Internet performance (e.g., [15,16]), including the performance of web servers and web services (e.g., [20,21]). However, we are unaware of performance tests for the Comet or WebSockets technologies we discuss below.

Last, researchers and practitioners in the WWW community have been very active in the development of web-based networking techniques. This work appears in the standards themselves (e.g., [26]), but aside from the numerous practical reports on these approaches (e.g., [4]), there is little guidance available for developers about how well web-based networking will work for groupware.

Real-Time Groupware Network Requirements

As previous researchers have noted, real-time groupware is different from other types of distributed systems in that it sends several types of messages with varying quality-of-service requirements [6,7]. As a way of contextualizing the performance tests described below, we here consider five canonical types of interaction that must be supported by groupware, and specify network requirements for each type.

- *Sequential turns and moves*. In card games and board games, or systems with floor control, people perform single actions in turns. Messages to indicate the actions are generally small, and the turns happen infrequently (perhaps a maximum of one per second). The data requirements do not change as more people join, since only one person acts at a time.
- *Text chat*. Text-based communication is a common part of many groupware systems. The data requirements of chat are generally characterized by a tradeoff between message rate and size: if chat is sent character by character, several small messages per second are required (e.g., 5/sec); if the system only sends larger blocks of text, then a smaller number of larger messages are needed. In addition, there can be a variable number of people in the chat session, from two to dozens (e.g., in Internet Relay Chat). At the server, text can either be delivered to clients as it arrives, or can be aggregated and sent using a timer.
- *Mouse movements*. Many kinds of groupware systems use mouse movement as an awareness cue – this is the basis for telepointers and for intermediate-state updates of actions such as lines in a shared drawing application. Mouse-position messages are small, but since mouse movement is difficult to predict, these systems require a high message rate (e.g., 20-30 updates per second for smooth telepointer movement). Various visual techniques (such as motion blur) can improve the appearance of streams where updates are less frequent, but the best-case scenario is to receive position information in real time.
- *Avatar movement*. For several kinds of games, such as first-person shooters and online role-playing games, a large fraction of the messages in the system are position updates for player avatars. Although each message is relatively small, there can be many players in the environment. To reduce network requirements, most

games use interpolation and extrapolation techniques that allow positions to be calculated even at a lower update rate. Updates from several players are aggregated at the server, leading to larger messages that are sent less frequently than for telepointers.

- *Audio and video*. Conferencing applications can make substantial demands on network infrastructure, depending on the quality of the audio or video signal. Although web applications cannot currently access devices such as microphones and webcams, future HTML standards may allow this kind of interaction [27]. Data rates for common audio and video standards range from as little as 20 Kbps for a single voice stream to about 2000 Kbps for 720p video using H.264, and even higher rates for higher-quality video [24].

Using these building blocks, we can list several different canonical types of groupware. Note that the requirements described below are not based on a rigorous analysis of existing systems, but rather are estimates of reasonable values so that we can later assess the types of groupware that web-based networking can support.

- *Card and board games*. Turn-based games have only minimal network requirements (a few bytes every few seconds). In some games, telepointers are also useful, which requires a higher message rate (see below).
- *Chat rooms*. Text-based communication is lightweight, although requirements increase as more people join the conversation. With two participants, a chat-room application would need to send no more than a few dozen bytes per second, and does not need a high update rate; larger rooms such as IRC may need to send hundreds of bytes per second (but all incoming messages can still be aggregated into a low update rate).
- *Shared workspaces*. Groupware systems such as shared whiteboards or group design environments involve manipulable objects in the shared workspace. The requirements for these systems are similar to those for mouse movement (and telepointers are almost always used in shared workspaces), but additional messages also need to be sent for the transactions in the workspace (e.g., drawing a line or moving an object). With fewer than ten people, these applications might need to send a few hundred bytes per second, with an update rate of 25/sec.
- *First-person shooter (FPS) games*. These games provide an avatar-based virtual environment where multiple people can interact. The main requirement for FPS games is that position updates must arrive frequently in order to improve the accuracy of avatar locations. A typical FPS might attempt to send updates at a rate of 20/sec., with message sizes dependent on the number of players.
- *Videoconferencing*. Although data rates vary, we will consider an example audio/video application that requires fast updates for audio (at 30-40/sec.) and large messages (e.g., 500 bytes per message), with low latency and jitter.

Additional Requirements

Although we focus on message rate, message size, and network delay, there are several other requirements that can be considered for a groupware system. We return to these later in the paper, and briefly assess whether the Web technologies can support these requirements as well.

- *Variable quality of service (QoS)*. Different messages in groupware have different delivery requirements in terms of allowable latency, reliability, and ordering [6,7]. Providing these capabilities often requires detailed control over the underlying network – for example, being able to select UDP transport instead of TCP when messages do not need guaranteed reliability.
- *Adaptive displays for different devices.* When groupware runs on heterogeneous devices (e.g., desktops, large displays, and mobiles), the interface and the architecture needs to adapt to provide the most appropriate data and presentation to each participant [25]. For example, different video sizes or frame rates might be delivered to different people based on their display capabilities.
- *Synchronization between streams.* When a groupware system uses multiple modes of interaction (e.g., voice and gestural communication), synchronization issues become important (e.g., making deictic gestures occur at the same time as spoken references). This requires fine-grained control of network use at the application level.
- *Development support.* Although not a networking issue, developers need support for building and deploying groupware systems. Support tools such as IDEs, debuggers, and toolkits (both for networking and interfaces) are now common for stand-alone applications.
- *Graphics performance.* Many interactive multi-user systems are now graphics-intensive; in particular, online games require both high-performance networking and a graphics environment that can provide fast screen updates and sophisticated rendering.
- *Cross-platform support*. A major deployment issue for groupware designers is ensuring that software will run on all platforms needed by the system users. This issue is common in all software development, but is relevant for groupware, since the system will often be used by multiple people working from different platforms.

WEB-BASED NETWORKING APPROACHES

The networking infrastructure of the WWW was not designed to support highly interactive applications or synchronous groupware, but rather to provide simple page content from a server to a client. However, web technologies have gradually been adapted to support more interactive capabilities. In the following sections we review the capabilities of three main approaches to providing networking functionality in the browser: re-purposing existing network mechanisms, integration of socket techniques into the browser, and plug-ins.

Re-Purposing

The Web is already a distributed and networked system, and so browsers already contain several mechanisms for

network communication. One possibility for supporting real-time groupware is to make use of these existing capabilities, based on HTTP, AJAX, and Comet.

HTTP

The HyperText Transfer Protocol (HTTP) is an application-level communication protocol built on top of TCP/IP. HTTP is the original mechanism for getting information from a web server. In the early days of the Web, only static documents and images were served to browsers, but several additions (e.g., CGI, Java Servlets, Active Server Pages) were developed to allow provision of dynamic content.

In HTTP-based communication, a browser initiates a connection to a web server using a TCP socket, and sends a request to the server. The server processes the request and replies in an HTTP response. The content can consist of HTML, JavaScript, image content, or other data types. The HTTP protocol allows any type of content, as long as the consuming application supports those types. Standard HTTP is not sufficient for real-time groupware, however, because browsers typically reload the entire page when receiving an HTTP message, which limits the update rate to less than one frame per second.

AJAX and XMLHttpRequest (XHR)

XHR was devised to let browsers send and retrieve data from web servers without having to reload the page. With XHR, the underlying HTTP and network mechanics are identical to retrieving a web page. However, rather than using a page URL, the developer accesses an XHR JavaScript object that allows browsers to process a server response as an asynchronous callback. Since JavaScript is the underlying technology, user interface events can trigger functions that load and display dynamic content.

To send data to a web server, a JavaScript function uses the XHR object to create an HTTP request. To retrieve data from the server, a JavaScript function uses XHR to ask the server for data. If data needs to be retrieved continuously (e.g., for telepointers), the JavaScript function needs to continuously poll the server for updates.

AJAX has proven successful for providing dynamic content without reloading pages. However, sending updates from the browser to the server, and polling for new data, requires that a new socket connection and a new HTTP message be created for each update, which is resource-intensive.

Comet

The resource problem of AJAX is partially solved with a set of technologies collectively called Comet. They allow a server to push data to the browser ('server push') without requiring a new connection for each update (note, however, that all communication from browser to server must still use XHR as described above). There are three main Comet techniques: long polling, XHR streaming, and iframe streaming. There are toolkits that package these techniques, (e.g., Ajax Push Engine, www.ape-project.org/; or Google App Engine, code.google.com/appengine), but here we focus on the underlying technologies individually.

Long polling allows the server to update a browser when it has new data to send, which frees the browser from having to poll the server continuously. Long polling reduces the number of connections and HTTP messages by only sending a response when the server has new data. The connection between browser and server sits idle either until it times out or until the server has data to send. When data is received or a timeout terminates the connection, the browser reconnects and waits for the next update. The network usage of long polling is more efficient than polling when data is sent infrequently (but needs to be received quickly by the browser). However, when data is constantly updated, as with real-time groupware, the performance of long polling degenerates to standard polling.

XHR Multipart Streaming takes advantage of an HTTP content type called 'multipart' that allows a web server to send content to a browser in multiple pieces. This type was designed for large messages (e.g., images), but can also be exploited in an XHR response by sending a complete message to the browser in each part, keeping the connection open for the next message that needs to be sent. Essentially the browser is tricked into keeping the socket connection open, with the server sending each update as a part. Using a single socket connection also reduces the number of HTTP headers that need to be sent for requests and responses.

In XHR multipart streaming, the browser makes an initial request to the server. The server indicates the content type as 'multipart', and the browser keeps the connection open as it waits to receive the parts of the HTTP response. Whenever the server needs to pass data to the browser, it sends a message as a response part. Finally, when the application is ready to be closed, the server sends the final response part. One limitation to XHR streaming is that in some cases, the browser stores each part of a multi-part message in memory (control over this functionality is part of the API, but our testing showed that actual behaviour is browser-dependent). To reduce memory load when using XHR multipart streaming for other purposes, the multi-part message is often closed and reopened periodically.

XHR Iframe Streaming also maintains a single connection to the server, but uses a hidden iframe as the message. The 'source' attribute is set to a CGI, Java Servlet, or Active Server Page that streams dynamically-generated JavaScript to the browser. The initial response provides the appropriate HTML tags, and then each message sent includes a <script> tag with a JavaScript function call that passes the real message as a parameter. The browser page implements this function, and processes each message as it arrives. As with XHR-multipart, the browser stores each part of the 'frame set', meaning that this technique also requires that the connection be reset periodically.

Adding Socket Functionality to the Browser

The Comet techniques described above re-purpose existing mechanisms that were designed for uses other than real-time communication. They therefore require the application

programmer to understand the details of the original mechanism, introducing additional layers of complexity to the system. An alternate approach is to incorporate true real-time networking mechanisms into the browser itself, an approach realized in the WebSocket standard of HTML5.

WebSockets are bidirectional, full-duplex communication channels based on TCP sockets, and are part of the HTML5 standard developed by the W3C (dev.w3.org/html5/spec/). WebSockets are built into browsers that support the HTML5 standard (e.g., Google Chrome 5, Mozilla Firefox 4, and Apple Safari 5).

The WebSocket standard provides an API that is accessible from within JavaScript, allowing developers to open a socket to a server, and send and receive data. JavaScript functions are automatically called when data is received. The WebSocket API provides only basic functionality, and does not provide the same degree of control over the socket that is offered by many stand-alone languages.

Plug-ins

For most of the history of the Web, the only way to send and receive real-time data was through plug-ins such as Java Applets or Flash applications. A plug-in is a software module that adds a specific capability to a larger system [23]. Web browsers have long used plug-ins for a variety of purposes, including displaying proprietary document formats, playing video, or accessing devices (such as cameras) that are outside the standard Web security model.

There are several browser plug-ins that allow the development and deployment of real-time groupware. These are typically based on an existing programming language and application framework, and adapt these existing facilities for use in the browser. Three common plug-ins that can be used for groupware are Java applets (based on the Java language and toolkit), Adobe Flash/Shockwave (based on Flex or ActionScript), and Microsoft Silverlight (based on .Net). Applications developed with these tools typically run in a byte-code environment that allows them to execute faster than interpreted JavaScript.

All of these plug-ins provide full-featured environments with extensive support for dealing with user input, graphics, interfaces, networking, and threading. However, security restrictions in the browser can mean that developers must work with a restricted API and limited architectural models for networking (e.g., unsigned Java applets can only make socket connections to the web server). Despite these limitations, plug-ins provide an execution environment that is essentially equivalent to that of a stand-alone application.

However, many developers and Internet application companies have recently rejected the plug-in approach in favour of standards-based technologies. There are three main problems with plug-ins:

- *Installation and availability*. In many web browsers, the plug-ins needed to run a groupware application may be missing or unavailable. This means that groupware developers have no idea whether their applications will run correctly when deployed, and means that users may have to take time to find and install the necessary plug-ins. This problem is particularly acute when planning for devices such as smartphones or tablets, since the browsers on many of these devices do not allow plug-ins at all.
- *Dependence on closed technology*. Plug-ins are often proprietary, reducing a developer's control over what capabilities are available and how their groupware system executes. As Steve Jobs states regarding Apple's decision to not include Flash in the iOS browser, "letting a third-party layer of software come between the platform and the developer ultimately results in sub-standard apps and hinders the enhancement and progress of the platform. If developers grow dependent on third party development libraries and tools, they can only take advantage of platform enhancements if and when the third party chooses to adopt the new features." [12].
- *Security*. Plug-ins are often themselves large and complex software systems, and can be vulnerable to attacks and security problems. For example, Adobe's Flash plug-in was the second most attacked system in 2009 [19], and many security experts advise not using Flash when visiting untrusted websites. Users can even install additional add-ons to stop plug-ins like Flash, Java, and Silverlight from executing.

WEB NETWORKING BASELINE PERFORMANCE STUDY

The goals of our performance study were to determine whether real-time groupware can be supported with web-based networking, which of the canonical types of groupware each approach can support, and how web-based networking approaches compare to a plug-in solution (as exemplified by Java applets).

Methods

We implemented web applications using each of the technologies described above, and carried out a series of performance test with each application. We tested the applications in three different network environments:

- *LAN* (Fast Ethernet, 3 hops to server, ping ≈ 0ms; browser on 32-bit Windows 7 Core 2 Duo CPU);
- *MAN* (ADSL, 5 Mbps down, 64 Kbps up, 4 km, 9 hops to server, ping ≈ 96ms; browser on 64-bit Windows 7 Core i7 CPU);
- *WAN* (CA*NET university network, 2700 km, 11 hops to server, ping ≈ 48ms; browser on Windows XP Pro, Core 2 Duo CPU).

The servers ran on a 32-bit Windows 7 PC with a Core 2 Duo CPU. All tests were carried out with the Firefox 4.0b7 browser, which performed best in pilot tests for all techniques. Implementation details for each technology are:

- *Long Polling, XHR Multipart Streaming, and Iframe streaming*. On the browser, we wrote JavaScript code in a standard HTML document to send and receive test data according to each technology. The server was a Java

servlet running under Tomcat 6.0; the servlet sent and received messages based on requests from the browser.

- *WebSockets*. We wrote JavaScript code in an HTML document to connect a WebSocket and send and receive data. The server was a C# implementation of the WebSocket specification, and sent and received data according to requests sent from the browser.
- *Java Applets*. The applets and the Java server were built using Java SE 6.0; the server ran as a standalone program.

We carried out two main assessments: tests of maximum message rates with a standard 500-byte message, and tests of the effect of increasing message size on message rate.

Results

Network Overhead

We inspected packets sent using each technology to determine how much additional information (in addition to the message payload and the standard TCP/IP headers) was being sent. We found that the XHR techniques add additional headers (multipart adds 40 bytes, iframe adds 38 bytes) to each packet. For XHR-multipart, the extra headers consist of the content-type and a boundary marker at the end of the part; for XHR-iframe, the overhead consists of HTML <script> tags with an embedded JavaScript call to handle the message contents. No added headers were seen with WebSockets or Java Applets.

Additional network overhead is also incurred when establishing connections to the server. XHR-multipart and iframe require approximately 700 bytes to initiate a connection to the server and this connection needs to be reset periodically to flush the response text from memory. WebSockets and Java sockets do not require this overhead.

Message Send Rate: Browser to Server

The rate at which the browser can send messages to the server is the maximum rate at which a groupware client can update others in the collaborative session. As discussed above, groupware applications need to send updates at a maximum rate of about 25 messages per second.

We tested each technology by setting up the browser to send 10,000 messages as quickly as possible to the server. We performed 10 of these trials for each technology to get a mean send rate. Messages included a sequence number and a 500-byte dummy payload (this size is used to prevent the system from aggregating messages into a single packet, since we were not able to turn off Nagle's algorithm for the web-based mechanisms). We checked to ensure that all messages were correctly received.

Note that the test applications were not performing any other tasks (such as processing input devices, receiving incoming messages, or drawing graphics to the screen); all of these additional tasks would reduce the message rate somewhat (this issue is addressed below when we discuss example applications). Results are shown in Figure 1. Note that all Comet technologies use the same upload mechanism – an XHR object in an HTTP POST request; therefore, there are only three technologies in this test.

Two of the three approaches (WebSockets and Java Applets) were able to send messages at a rate well above what would be required in a real-time groupware system (i.e., 25 messages/sec. or less). The lower performance on the MAN setting does not indicate a limitation of the technologies, but rather shows that the data used all the available bandwidth of the connection. The performance of XHR sending in wide-area applications, however, could restrict this technology for groupware that requires a high update rate from each client. Our tests show that smaller messages see an increased XHR send rate, but do not dramatically change performance.

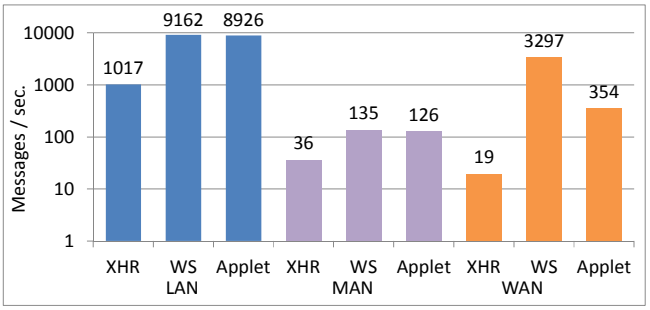

Figure 1. Browser to server message rate (500 bytes).

Message Receive Rate: Server to Browser

The server-to-browser send rate is the rate at which a groupware client can receive updates from others in the session. Again, the maximum required rate is about 30 messages per second or less (per collaborator). If the server aggregates these updates into a single message, then the required rate will be unchanged, but the message will be larger since the payload will include updates from several other participants. If the server passes on all updates without aggregating, then the required rate is multiplied by the number of participants in the session (however, this is unlikely since aggregation is a more efficient strategy).

All technologies other than Long Polling were able to receive messages at a higher rate than what would be required for real-time groupware (Figure 2), in all network settings. In the MAN setting, the faster technologies are all able to utilize the full bandwidth of the connection (accounting for the similar performance).

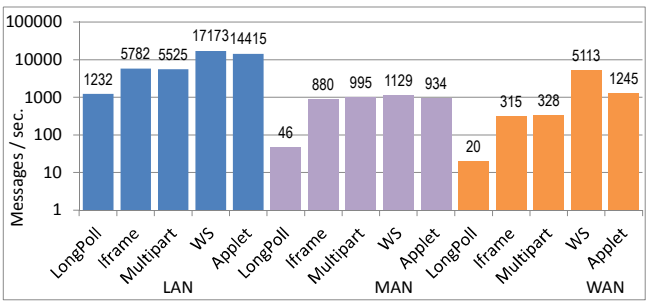

Figure 2. Server to browser message rate (500 bytes).

Maximum Receive Rate at Increasing Message Sizes

Some applications such as screen sharing and video conferencing can require both a high message rate and large messages. In our tests, we assessed the maximum message rate for each technology as message size increased.

As can be seen in Figure 3, there are substantial differences between the technologies, particularly with the smaller message sizes that are likely to be used in a groupware system. Nevertheless, all technologies are able to maintain a high message rate (more than 30 messages per second for all methods but Polling) even at message payloads of 1000 bytes. Surprisingly, WebSockets outperformed Java Applets in the WAN setting, by a wide margin (however, this result must be confirmed by further testing).

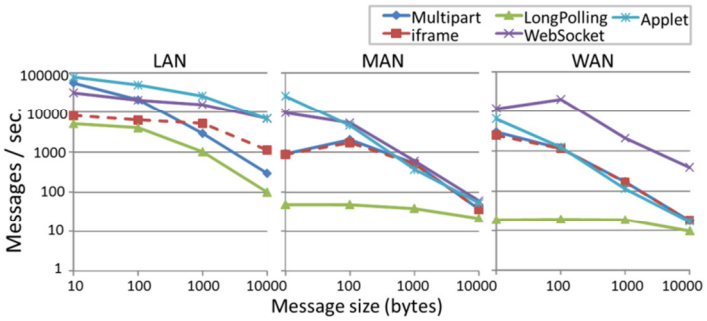

Figure 3. Max. receive rate at different message sizes.

REALISTIC GROUPWARE APPLICATIONS

To further test two technologies (Comet and WebSockets) in a real-world setting, we developed two example groupware applications. Developing real applications required that we go beyond the networking-only code used in the performance tests above. Interactive applications require user input, a user interface, and graphics output if the system provides a shared workspace.

Browser-based graphics and user-interface toolkits are still less mature than what is available for plug-in or stand-alone applications, but these tools are beginning to appear (for example, the Google Web Toolkit [9] provides support for many aspects of developing browser-based applications).

In addition, new tools are beginning to appear as part of web standards. The HTML5 Canvas element provides a substantial improvement in how browsers provide 2D graphics, and has allowed a port of the visual Processing language to JavaScript (processingjs.org). Processing.js provides support for capturing user input and drawing to the screen, and was used to develop both of the example applications described below.

Puzzle Game (using XHR-Multipart)

We wrote a simple multi-player puzzle game, which required both browser and server programs (Figure 5). The browser portion was written in Processing.js and JavaScript. Processing.js (which uses the HTML5 Canvas) was used to draw telepointers, and JavaScript was used to handle all communication with the web server. When the browser loaded the game web page, a connection was initiated to the server to receive messages via XHR multipart streaming. Mouse locations were encapsulated as a telepointer object and sent as a JSON-encoded POST parameter every 40 milliseconds using XHR send. If no mouse movement occurred during a 40-millisecond interval, no mouse event was sent to the server.

A Java Servlet processed messages on the server. The XHR streaming connection was reset every 30 seconds. When the server disconnected at the 30-second interval, the browser re-initiated the connection. The servlet pushed data to the browser every 40 milliseconds. Messages from browsers were stored using a mailbox model where each connected client had a mailbox containing messages it should receive. When the server received a telepointer or object-move message from a client, the message was put into all other clients' mailboxes. When sending to the browser, the servlet aggregated all mailbox messages for the client into a single message.

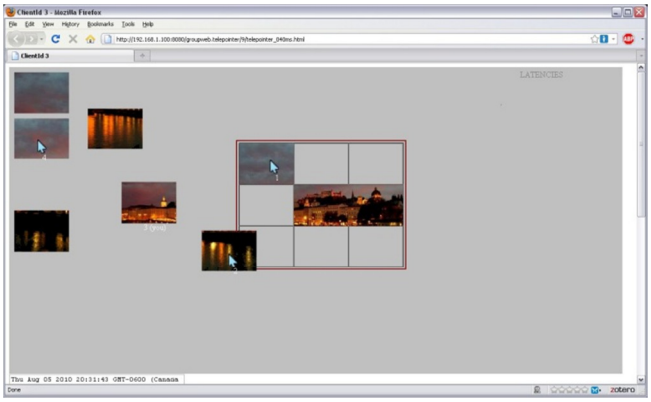

Figure 5. XHR-multipart puzzle game with four users.

Shared Drawing Editor (using WebSockets)

A group drawing program was built as a second example in order to demonstrate the use of WebSockets (Figure 6). The browser side of the application was also written in JavaScript (for network communication) and Proccesing.js (for graphics and user input).

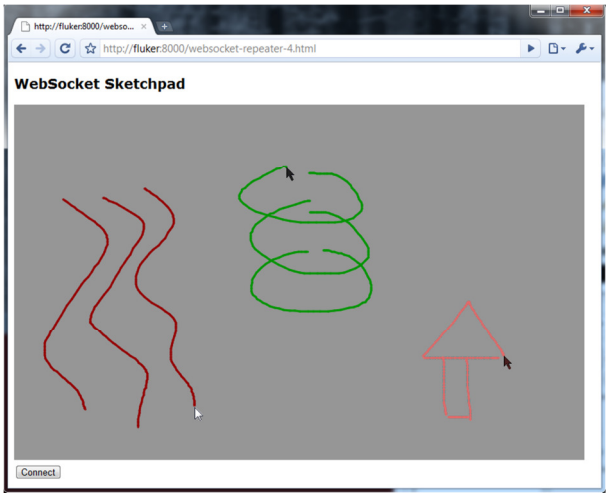

Figure 6. WebSockets drawing editor with three users.

The server program was written in C#, and acted as a message-passing repeater to broadcast incoming messages to all clients (although with rate control). Updates were sent to the server on every mouse interrupt (which handled both pointer movement and line drawing). All messages were sent as text, with telepointer updates and new line-segment messages sent separately. The server sent updates every 40ms, aggregating messages that arrived during that period.

Evaluation of the example applications

Our goals in evaluating the example applications were to determine whether real-time interaction can successfully be supported using web-based networking, whether the throughput results hold in real groupware systems, and whether the other necessary elements of a visual-workspace application (e.g., graphics, user input) will perform well enough in a browser-based application to meet the demands of real-time collaboration.

We assessed latency, jitter, and subjective performance in the three network contexts described above (LAN, MAN, and WAN). Latency was calculated based on a repeated series of round-trip times between the browser and the server, using the actual messaging that would be used to communicate data in the application. Jitter was calculated using these same messages, and is reported as the mean divergence from the average latency. Subjective usability was determined through simple user tests where we and others in our labs used the applications, and looked for episodes of lag, non-smooth movement, or other artifacts in the telepointer motion. Tests used the computers and networks described above; results are shown in Table 1. In general, latencies increase with distance, and are considerably higher than the ping times, indicating that the web-based technologies are adding overhead to the communication. WebSocket latencies are consistently low enough to support even lag-sensitive groupware such as real-time games; XHR-multipart showed longer delays, implying that this technology could not support the highest level of requirement for real-time interaction.

We note that real-world Internet latencies are highly variable, and further testing is required to provide a clearer picture of these techniques' delay characteristics. If these values are confirmed, however, they show a substantial difference between WebSocket and Comet approaches.

Table 1. Latency and jitter for example applications.

	XHR-Multipart			WebSockets		
	LAN	MAN	WAN	LAN	MAN	WAN
Mean latency (ms)	67.8	121.2	185.7	11.6	55.8	86.5
Mean jitter (ms)	13.7	6.6	9.5	8.5	7.1	6.5

For our subjective tests, we evaluated the smoothness and lag of telepointers, using two, four, and seven participants. The puzzle game was deployed in a laboratory of MacBook laptops on a dedicated LAN; all machines used Firefox 3.5. Participants were asked to move their cursor in a recognizable pattern (e.g., in a circle) so that motion and lag could be perceived. All participants agreed that there was no perceivable jitter or lag in the telepointer motion, and that there was no perceivable difference between two, four, or seven users.

For the wide-area tests of the puzzle game, we connected three computers at the three network sites (one across the city on the MAN; one across the country on the WAN). No noticeable problems with the motion of telepointers or pieces occurred during the test, and the participants were easily able to work together to complete the puzzle. The performance of the game over the real-world Internet was not perceptibly different from its performance during our LAN tests, and the user experience was easily comparable to that of a stand-alone groupware system.

These usability results help to confirm that the network performance results presented above do in fact translate to effective real-world groupware systems.

DISCUSSION

In this section we summarize our main results, address issues of real-world use of Web technologies for real-time groupware, provide several recommendations for groupware designers, and outline avenues for further work.

Summary of Results

Results of the baseline performance tests, latency and jitter tests, and usability evaluations indicate that web-based networking can successfully be used for many types of real-time groupware. In the throughput test, all approaches were able to maintain high message rates even with large messages. These performance results were consistent across LAN and MAN settings; however, the variance in the WAN tests suggests that XHR-based approaches may be limited for groupware that requires a high browser-to-server update rate. Latency tests suggest that WebSocket-based systems can support most groupware requirements; Comet-based approaches, however, may present greater limitations for groupware with strict lag requirements.

Our tests suggest that for most groupware types, web-based networking is not a major limiting factor in the speed and responsiveness of real-time groupware in the browser. A caveat in this conclusion, discussed further below, is that deploying real-time groupware on the Internet often requires more detailed control over application-level networking than what is available with web technologies, and these requirements mean that the highest requirement levels of distributed groupware (e.g., first-person shooter games or videoconferencing) may not yet be feasible.

Deployment and Development Issues

Our studies suggest that Web technologies perform well in terms of basic messaging rates; the next step is to determine what these technologies can offer for the other groupware requirements identified earlier.

- *QoS control*. This is an area where Web technologies are considerably less mature than stand-alone groupware approaches, providing essentially no support for application-level network control. All of the web-based

networking technologies use only TCP transport, and allow very little control over issues such as routing, maximum packet sizes, aggregation, or timeouts. The restriction to TCP also means that there is no opportunity to use less reliable transports such as UDP, which are better suited to awareness messages such as telepointers [6]. For example, the reliability guarantees of TCP mean that clients must wait for retransmission of a lost packet, even though that information (e.g., a single telepointer update) may not have been critical to the interaction.

- *Graphics capabilities*. Browser-based graphics is still far behind stand-alone applications, but the development of tools such as the HTML5 Canvas and web ports of OpenGL (e.g., en.wikipedia.org/wiki/WebGL) mean that Web-based graphics could soon approach the performance of plug-in technologies. Performance tests of these aspects of web-based applications is a clear area for future work.
- *Development environments*. It is still more complex to develop a groupware application using Web technologies than it is with more established approaches. Several issues contribute to this disparity: for example, the immaturity of WebSockets means that documentation and examples can be difficult to find; there are few environments for development; and the nature of Comet technologies (as repurposed capabilities originally designed for other uses) makes it more difficult to understand, design, and debug applications. There is a strong need for better tools in this area – e.g., groupware toolkits that use Web technologies, and development environments for Web applications.
- *Cross-platform deployment.* One of the main advantages of a standards-based Web approach is that cross-browser and cross-platform deployment should be made considerably easier. This is still likely to happen, but the current state is somewhat more chaotic: not all browsers support all the technologies, and implementations (and performance) can differ widely across browsers and platforms. For example, Internet Explorer 8 does not support XHR-multipart, and WebSockets are only implemented in Firefox 4, Safari 5, and Chrome 5. These differences will become less of a problem as browsers adopt HTML5, but the problem will remain at least for the short term. One capability that could improve this situation is that of falling back from one technology to the next, in order to improve a system's robustness. That is, a groupware system could try to use WebSockets (the best performing technology we tested), and then fall back to XHR-multipart if WebSockets are not supported, and then fall back to iframe for Internet Explorer. This adds complexity to the application, but this is exactly the kind of capability that could be built into a toolkit, and made invisible (or at least less painful) for the developer.
- *Access to devices and file systems.* The current security model of the Web prevents web pages from accessing any devices or files outside a very narrow sandbox. This means that groupware applications such as video or audio-conferencing are not currently possible using the Web networking technologies discussed here. However, it is likely that these restrictions will be relaxed in future HTML standards as more capabilities are added to browsers – for example, there is already a draft for a <device> element in HTML for access to webcams [27].
- *Real-world performance testing.* We tested applications in three different network environments, but more work needs to be done to test web-based networking in real-world situations – for example, situations where bandwidth is limited, where traffic patterns change, and where loss and jitter are common. As mentioned above, the use of TCP transport means that Web-based groupware may be much more susceptible to latency problems than systems that can use unreliable transports like UDP [7].

Recommendations for Groupware Developers

Our work can provide several lessons and guidelines for groupware developers. The main and most obvious recommendation is that developers should start to consider the browser as a legitimate vehicle for deploying real-time groupware – not just asynchronous or semi-synchronous systems. Our results suggest that Web technologies can support a wide variety of network requirements, including highly interactive workspaces and systems for large groups. The capabilities of these technologies are not well known in the CSCW community, but the good network performance seen in our studies suggests that a new space for deployment of real-time groupware is now available.

Second, our results show that WebSockets are the best-performing of the standards-based web technologies. In addition, they are simple to work with (compared with the Comet techniques) and have a conceptual model that groupware developers are already familiar with. Although not all browsers support WebSockets, this is likely the technology that will eventually become the standard for groupware networking in the browser.

Last, our experiences also recommend caution given the early state of some of these technologies, and the lack of powerful development tools. Although this situation will undoubtedly improve, developing and testing large-scale groupware applications is likely to be more difficult than an equivalent stand-alone or plug-in solution.

CONCLUSIONS AND FUTURE WORK

Standard web browsers are increasingly becoming a platform for delivering rich groupware applications. However, the traditional way that these applications are deployed – using browser plug-ins – presents several problems that have led many companies and developers to look for other solutions such as AJAX, Comet, and WebSockets. These web-based networking approaches are capable of providing real-time interaction, but there is little information available about whether they can support real-time groupware. To establish baselines for the performance of web-based networking, we tested the performance of two recent approaches (Comet techniques and WebSockets) and

compared them to a plug-in solution, Java Applets. We found that in terms of message rates and message sizes, web-based networking approaches were able to maintain throughput that would support a wide variety of real-time groupware. We also developed two example applications (using XHR-multipart and WebSockets) and showed that they performed well in three real-world network settings. Our work shows that developers can begin to consider the web browser as a legitimate vehicle for deploying a wide range of interactive real-time groupware.

These studies suggest three main directions for further research. First, we plan to further test these web technologies in realistic applications on the real-world Internet, and determine how well they deal with issues of lag and restricted bandwidth in everyday use. Second, we will extend our tests to browsers on other platforms, particularly phones and other mobile devices. Third, we plan to carry out performance tests for other parts of a browser-based groupware system (such as graphics). Last, over the longer term we plan to develop a groupware toolkit based on WebSockets that will simplify the development of web-based real-time groupware. The toolkit will provide groupware developers with the same types of support that are already common for stand-alone groupware, such as session management support, awareness widgets, shared data structures, and debugging tools.

ACKNOWLEDGMENTS

This work was supported by NSERC, the SurfNet research network, and the GRAND NCE network.

REFERENCES

1. Bentley, R., Horstmann, T., and Trevor, J., The World Wide Web as Enabling Technology for CSCW: The Case of BSCW, *CSCW*, 6, 2-3, 1997, 111-134.
2. Buszko, D., Lee, W., and Helal, A. Decentralized ad-hoc groupware API and framework for mobile collaboration. *Proc. Group 2001*, 5-14.
3. Chung, G., Dewan, P., and Rajaram, S. Generic and composable latecomer accommodation service for centralized shared systems. *Proc. EHCI 1998*, 129-148.
4. Crane, D. and McCarthy, P., *Comet and Reverse Ajax: the Next-Generation Ajax 2.0*. Apress, 2008.
5. de Alwis, B., Gutwin, C., and Greenberg, S., GT/SD: Performance and simplicity in a groupware toolkit, *Proc. EICS 2009*, 265–274.
6. Dyck, J., Gutwin, C., Subramanian, S., Fedak, C., High-performance telepointers. *Proc. CSCW 2004*, 172-181.
7. Dyck, J., Gutwin, C., Graham, T., and Pinelle, D., Beyond the LAN: techniques from network games for improving groupware performance. *Proc. GROUP 2007*, 291-300.
8. Google Inc., *Laying the Foundation for a New Google Docs*, googleenterprise.blogspot.com/2010/04/laying-foundation-for-new-google-docs.html (May 17, 2010).
9. Google Inc., *Google Web Toolkit Overview*. code.google.com/webtoolkit/ (July 31, 2010).
10. Hall, R., Mathur, A., Jahanian, F., Prakash, A., and Rassmussen, C. Corona: a communication service for scalable, reliable group collaboration systems. *Proc. CSCW 1996*, 140-149.
11. Ionescu, M., and I. Marsic. Latecomer and Crash Recovery Support in Fault Tolerant Groupware. *IEEE Distributed Systems Online*, 2, 7, 2001.
12. Jobs, S., *Thoughts on Flash*, www.apple.com/hotnews/thoughts-on-flash/ (May 17, 2010).
13. Mills, E., *Adobe Flash policy is risky*, news.cnet.com/8301-27080_3-10396326-245.html (July 31, 2010).
14. Morgan, S., and Wang, W., The Impact of Web 2.0 Developments on Real-Time Groupware, Proc. *IEEE Conference on Social Computing 2010*, 534-539.
15. Navarre, D., Palanque, P., Basnyat, S., Usability Service Continuation through Reconfiguration of Input and Output Devices in Safety Critical Interactive Systems, *Proc. SAFECOMP 2008*, LNCS 5219, 373-386.
16. Palmer, J., Web Site Usability, Design, and Performance Metrics, *Information Systems Research*, 13, 2, 151-167.
17. Roseman, M., and Greenberg, S., Building real-time groupware with GroupKit, a groupware toolkit. *ToCHI*, 3, 1, 1996, 66–106.
18. Russel, A., *Comet: Low Latency Data for the Browser*, alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/, retrieved July 31, 2010.
19. Symantec Inc., *Internet Security Threat Report: Volume XV: April 2010*, www4.symantec.com/Vrt/wl?tu_id=SUKX1271711282503126202, (July 31, 2010).
20. Tian, M., Voigt, T., Naumowicz,, T., Ritter, H., Schiller, J., Performance considerations for mobile web services, *Comp. Comm.*, 27, 11, 2004, 1097-1105.
21. Titchkosky, L., Arlitt, M., and Williamson, C. A performance comparison of dynamic Web technologies. *SIGMETRICS Perform. Eval. Rev.* 31, 3, 2003, 2-11.
22. Wang, W. Powermeeting: gwt-based synchronous groupware. *Proc. Hypertext 2008*, 251-252.
23. Wikipedia, *Plug-In (computing)*, en.wikipedia.org/wiki//Plug-in_%28computing%29 (July 15, 2010).
24. Wikipedia, Bit Rate, en.wikipedia.org/wiki/ /Bit_rate (Nov. 30, 2010)
25. Wolfe, C., Graham, T.C.N., Phillips, W.G., and Roy, B., Fiia: User-Centered Development of Adaptive Groupware Systems, *Proc. EICS 2009*, 275-284.
26. World-Wide Web Consortium, *The Web Sockets API*, http://www.w3.org/TR/websockets/ (July 31, 2010).
27. World-Wide Web Consortium, *HTML Device*, http://www.w3.org/TR/html-device/ (July 31, 2010).