A Flexible Architecture for Customizing Web Streams for Wireless Clients

Jesse Steinberg and Joseph Pasquale
University of California, San Diego
Department of Computer Science and Engineering
La Jolla, CA 92093 USA
(1) 858-534-8604
{jsteinbe,pasquale}@cs.ucsd.edu

***Abstract*—We present the design of the Web Stream Customizer Architecture, the basis for a middleware system that supports customization of Web content and streams. Our notion of customization is quite general, including functions such as filtering, remote buffering, remote caching, flow regulation (e.g., for video playback), compression, and encryption, all of which are especially valuable for wireless resource-limited clients, which are a primary target of support. The system is highly deployable, as it works entirely within the Web's common programming and communication frameworks, such as using HTTP and relying solely on standard proxy mechanisms, and therefore does not require any changes to existing Web servers and browsers.**

***Index Terms*—HTTP, Middleware, Proxy, Wireless, Mobile Code**

I. INTRODUCTION

While the popularity of wireless devices continues to increase, there remains a vast amount of content not designed for access by these devices. Rather, the more typical expectation is that access will be from desktop computers with relatively plentiful resources, such as large, high-resolution displays, large memories, fast processors, and reliable high-throughput network connections. One approach to addressing this problem is to dynamically customize content to meet the requirements of a particular device that does not meet these assumptions, as well as to allow it to adapt to changing network conditions. And since users of wireless devices may relocate at any time, this customization should take the user's location into account to increase performance.

Customization can provide significant benefits in terms of performance, reliability, security, and power consumption for activities such as wireless Web browsing. One of the most commonly discussed examples of improving performance is the use of remote filters that can reduce the sizes of objects being transferred over low-throughput connections. In many cases, objects may require filtering anyway in order to be handled by resource-limited devices, such as those that cannot store, process, and display HTML pages containing large high-resolution images (or simply large fonts that are too big for the display). The benefit of filtering remotely is that the amount of data being transferred over the lower-throughput wireless link is reduced, and the filtering can be done on a machine with much greater processing power, while reducing power consumption at the client.

The same arguments apply for reliability. Consider that a connection failure during a Web transaction could lead the user to be unsure of the outcome of that transaction (e.g., if the user clicks to pay for something, but does not get a response because of a disconnection, did the request make it to the server or not?). By storing the results of a transaction at a more reliable remote location, the user can easily check on its outcome once connectivity is reestablished.

Regarding security, the susceptibility of wireless networks to eavesdropping may lead an unwary user to expose information that they would otherwise prefer to keep private, such as their phone number and address, unless the sensitive data is encrypted before crossing the wireless link. Thus, whether for performance, reliability, or security reasons, the ability to act remotely is critical.

In many cases, it may be important that when remote customization is taking place, the location of the user be taken into account. For example, envision a scenario of a user who has a mobile device at work, accessing Web resources from any location on the company campus. Remote customization may conveniently take place on a PC in the user's office. When the user travels to meetings and conferences in other cities around the world, the user would like to use the device to access Web resources and customize the content in the same manner. Performance may suffer if the customization were to still be done in the user's office, since all of the HTTP transactions would have to go through the office PC. Ideally, the location of the customization should move to a location closer to the user.

The Web Stream Customizer Architecture is designed to improve the performance, reliability, and security of Web browsing for wireless devices, with support for user relocation. The main idea is the use of customization modules, called Customizers, which act remotely on behalf of a client. An early version of the design was presented in [29]. In this paper, we present a significantly improved design that supports new capabilities, including special-purpose (non-HTTP) stream communication and processing, and support for the movement of Customizers in response to user relocation. We show that the overhead of the system is low and tolerable, and that the gains of remote customization can be significant,

especially when transferring video streams that have high throughput and (soft) real-time response requirements.

Our system is designed based on the following goals:

Transparency: For ease of integration, the system is transparent to the operation of current Web servers, requiring no changes to their structure, and is compatible with standard Web browsers. The system is implemented using only existing Web mechanisms, such as HTTP and proxies.

Remote processing: The system provides the ability to act at a remote location relative to the client, e.g., beyond the wireless link for an untethered user, which is crucial to gaining significant benefits in performance, reliability and security. Intermediaries between the client and server provide remote locations for Customizer execution.

Adaptability: Content customization can be targeted for the particular device being used, and the customization can be adapted based on changing system/environment conditions, including fluctuations in network throughput, and resource availability on the client. To adapt dynamically to changing conditions, each Customizer actually has two points of control: the customization module acting remotely, and an assistant that is closely coupled with the client device.

Flexibility: Ideally, when a Customizer is loaded, it is executed at the location that achieves the maximal benefit in terms of performance, reliability, or security. It is also possible to have multiple Customizers that are simultaneously active, possibly at different locations, each handling requests to different Web sites. Allowing third-party servers as well as a personal server belonging to the user to host Customizers provides flexibility of location.

User relocation: If the user relocates, it should be possible for the Customizer to move to a new location, e.g., closer to the user, to improve performance. Rather than moving the Customizer in mid-execution by capturing, moving, and reestablishing its execution state, such as is done with mobile agents (based on a strong mobility model of movement), we take a simpler approach based on a limited form of mobile code in conjunction with a soft-state model of Customizer execution. In this model, moving a Customizer reduces to the case of reloading the Customizer on a new host. Furthermore, the use of a personal server provides a central, well-connected location for storing the Customizer, its configuration, and any long-term state, to facilitate Customizer reloading.

Practicality: The user should not have to rely solely on the provision of resources by third parties in order to run Customizers. In fact, we expect that Customizers will typically run on a personal server that is owned by the user, thus reducing dependence on third parties, making deployment simpler and more realistic.

The rest of the paper is organized as follows. We describe the architecture in Section II. In Section III, we present some example applications. In Section IV, we present the performance results of the implementation, including a regulated video buffer application. We review related work in Section V, and present conclusions in Section VI.

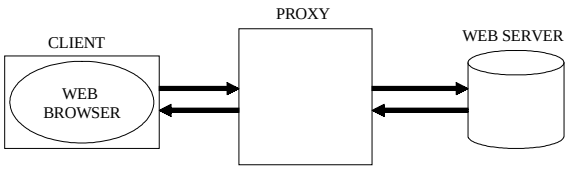

Fig. 1. Using a proxy to customize Web transactions.

II. The Web Stream Customizer Architecture

A common approach to remote Web customization is the use of HTTP proxies as intermediaries between the client and server, as shown in Figure 1. In this model, requests generated by the client are sent to the proxy instead of to the Web server. The proxy then forwards the request to the Web server. The Web server receives the request, processes it, and returns a response to the proxy. The proxy then has the opportunity to customize the response before it is returned to the client. This approach is transparent to Web servers since they see the proxy as a client, and is also transparent to clients since most popular Web browsers have a proxy mechanism that allows them to automatically forward their requests to a proxy.

In the Web Stream Customizer Architecture, the role of the proxy is expanded. First, there is the Customizer Server (CS), which provides an execution environment for running Customizers. A user typically will have multiple Customizers that are active, which depend on their function, e.g., filtering, compression, encryption, caching, etc., and which may apply only to specific sites, e.g., cnn.com, yahoo.com, etc. Figure 2 shows a client using multiple Customizers, each of which is running on a separate CS (including a special one designated as the PCMS, discussed below).

To support adaptability, each Customizer has an associated helper module called a Customizer Assistant (CA). The CA runs on a Client Integration Server (CIS), which tends to be located on or near the client device, and acts primarily as an extension of the browser (given that the browser code itself cannot be modified). Serving as a proxy for the browser, all of the browser's requests are sent to the CIS.

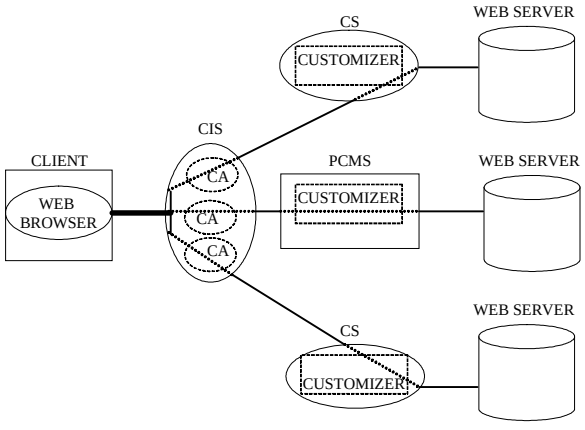

Fig. 2. A client using multiple Customizers.

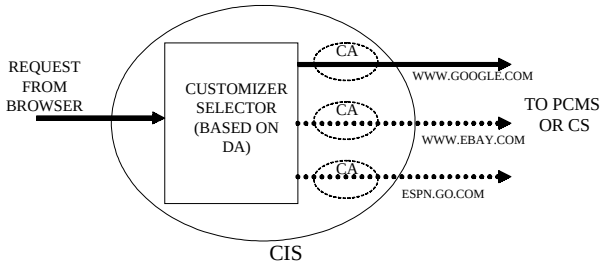

Fig. 3. Selecting a Customizer based on the DA (Domain of Applicability).

Thus, when a Customizer is being used, the request passes from the client to a CA (which runs on the CIS), then to a Customizer (which runs on a CS), and then to the Web server (and vice-versa for responses in the opposite direction). Despite the additional stops a request and response must make, we have found the resulting overhead to be insignificant relative to typical end-to-end Web request/response times without Customizers, as reported in [29], and summarized in Section IV.

Given its close coupling with the client, the CA is generally responsible for tasks that require knowledge of resource availability and system conditions at or near the client, which may then be communicated to the Customizer (e.g., such as relaying local system or network performance status so that the Customizer can adapt, to improve performance). In addition, the CA will also reverse data transformations done by the Customizer, such as for compression/decompression or encryption/decryption.

How does the CIS know which CA should be given a request from the browser (and consequently, to which Customizer the request should be passed to)? Associated with each Customizer is its *domain of applicability* (DA), which is a list of all of the sites for which the Customizer will act. When a Customizer is deployed (described below), its DA, along with its corresponding CA and the Customizer's current location, is downloaded to the CIS. The CIS then uses the DAs to perform the matching of requests to CAs and Customizers.

Figure 3 shows how HTTP requests are handled by the CIS. When the CIS gets a request from the browser, it first matches the URL of the request to the DA of all loaded Customizers. If there is a match with a Customizer, the request is passed to the corresponding CA, and then to the CS for that Customizer. The Customizer does not need to be explicitly loaded at that location, as described below.

Typically, a user will own a personal computer at their home or office (or at least have access to a machine at least as powerful on which they have an account), that has reliable Internet connectivity relative to the wireless PDAs we have been considering as clients. To facilitate user relocation and improve the practicality of our customization system, such a machine can be configured to serve as the user's Personal Customizer Management Server (PCMS). The PCMS is a storage place for the code modules of the user's Customizers, along with their preferences and state (collectively called a *Customizer Package*).

The PCMS can make dynamic decisions about where the Customizer code will run, based on the location of the user and the constraints of the Customizer (such as security restrictions), and it can carry out Customizer reloading without any direct involvement by the low-powered client and its possibly unreliable low-bandwidth wireless connection. Customizer reloading provides a measure of fault tolerance, because a Customizer may also be moved if there is a significant lapse in response time from the current CS on which the Customizer is running.

The PCMS contributes to practicality because it can also be used as a (user-controlled) location for actually running Customizers, which is especially useful if there is no available CS that is willing to host the Customizer. Figure 2 shows an instance of the complete Web Stream Customizer Architecture, with Customizers running on Customizer Servers and on a PCMS.

III. Applications

We are currently experimenting with a number of applications that focus on improving performance, reliability, or security. These are an Image Filter (as part of a Filter-Saver general application), a Transaction Recorder, an Encryptor, and a Flow-Regulating Buffer (for high-bandwidth streaming of video).

A. Remote Filtering

We have implemented an adaptive Image Filter Customizer that can modify the resolution, color-depth and compression rate of images. The CA measures the approximate throughput at the client, and relays it to the Customizer. This allows the Customizer to adapt its filtering to maintain consistent transfer times as throughput fluctuates. We are also experimenting with a more general Customizer, called the Filter-Saver, which can be used for a number of applications. As the name implies, it consists of two major components, a Filter and a Saver. The Filter reduces the amount of data in an HTTP response coming from the Web Server as it passes through the Customizer, and the Saver takes advantage of the storage available on the PCMS by saving original versions of the objects before they are filtered.

Storing the objects at the PCMS ensures that they will be available even if they are removed from the original source, and allows the objects to be retrieved quickly regardless of network problems between the PCMS and the original source. (In fact, the objects are available immediately if the user accesses them from the host on which they are running their PCMS).

In addition to filtering objects based on their data type, content-specific filtering can be done. For example, the Filter could remove the commentary from a product review site and return only the final score; research papers might be filtered so that only the pages containing the abstract, introduction, and bibliography are returned; scene change detection could be used to reduce the frame rates of videos.

B. *Transaction Recording for Reliability*

The Transaction Recorder addresses the problem of connection failures during a transaction by storing the results of transactions on the PCMS. The user can easily check on the outcome of the transaction once connectivity is reestablished via the Customizer's configuration page, which is accessible by the user from a Customizer control Web page provided by the CIS.

The Connection Smoother Customizer is an extension of the Transaction Recorder. It stores objects requested by the browser on the PCMS. If there is a short-term lapse in connectivity before the browser receives a stored object, the CA automatically repeats the request for that object, to mask the connection failure from the user. Since objects are only stored for short periods of time, this Customizer performs best when it runs on the PCMS, or when the CS has a cache for the data sent to the PCMS.

C. *Security*

The Selective Encryption Customizer encrypts sensitive data (e.g., before it crosses a wireless link), and the CA decrypts the data before it is passed to the browser. The Customizer can be configured to match sensitive text given by the user (such as their e-mail address, phone number, etc.) to the document, and, to increase performance, encrypts only those documents that contain sensitive data.

D. *Flow-Regulating Buffer for Streaming*

So far we have described applications that customize HTTP transactions. It is also possible to use Customizers for applications that use other protocols. To enable this, Customizers that are trusted by a CS are given privileged access to resources that are persistent across multiple HTTP transactions, including memory for data, hard disk storage, and threads of execution. The Customizer must also have network I/O privileges.

One non-HTTP application that especially benefits from the ability to reload a Customizer near the client is video streaming. For example, a wireless resource-limited device may not have enough memory to adequately buffer streams, whereas a Customizer running near the client can buffer the stream and periodically feed the client. The Customizer can also filter the stream to match the capabilities of the client device, or regulate the flow of data to the client device. Potential benefits include decreasing client buffering requirements, improving playback quality, reducing interruptions to playback, reducing the window of time that the playback is susceptible to WAN and server failures, overcoming the protocol limitations of Web servers, reducing restart latency, and reducing power consumption.

On the Web, video streams are typically initiated either from a link within a page that contains a URL that causes the browser to launch the video client, or by a page having a link to a metafile that contains information about the stream, and which is downloaded by the browser using HTTP and then passed to the appropriate streaming client. Customizers can be used to intercept video streams for filtering by using the initiating Web page or metafile as a hook.

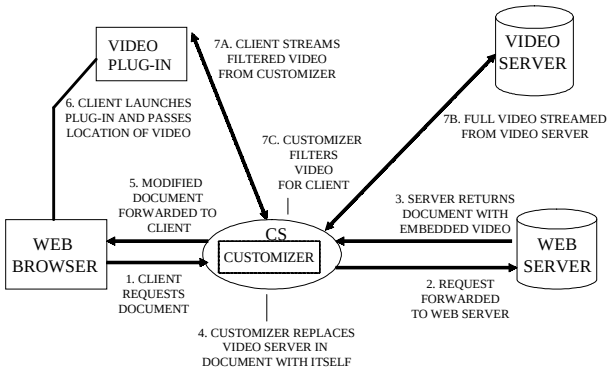

Fig. 4. Using Customizers to filter video streams.

When a Web server replies to a request with a page linking a stream, or a request for a metafile specifying the location of a video file, the Customizer modifies the location of the video stream. It replaces the identity of the source of the video with itself, so that the client' s video application tries to retrieve the video stream from the Customizer. The Customizer then acts as a proxy for the video streaming protocol. The Customizer appears to the video server to be the video client, while the client sees the Customizer as the video server. This is shown in Figure 4.

For customization that requires local action before being sent to the client, the Customizer can use a CA, and they can act in concert as a pipelined pair of proxies operating on the data. This approach allows Customizers to act on streams between the video client and server if it is able to implement the appropriate protocol.

It is also possible for Customizers to use their own video client, and even stream objects that were not initially set up for streaming by the content provider, by use of Customizer-specific helper-applications pre-installed at the client. This allows the Customizer to control the streaming more directly. In this case, when the Customizer gets a request for a video object or a metafile representing a video stream, instead of modifying the server directly in the document, the Customizer replaces the requested document in the response with a specialized metafile type associated with the Customizer-specific helper application. The specialized metafile will specify the Customizer as the video server. When the browser receives the specialized metafile, it will launch the Customizer-specific helper application, which will then act in concert with the Customizer to stream the data and display it to the user.

This mechanism can be used to create streams out of objects that would normally be downloaded fully by the browser. For example, suppose the user clicks on an HTTP request for an MPEG file. Since the HTTP protocol is being used, the standard browser behavior is to download the entire file and hand it to the appropriate helper application (in this case, a video player). However, if the Customizer responds with a specialized metafile that it generates on the fly, instead of the MPEG clip, the browser will open the Customizer-specific helper application associated with the metafile's extension (or the HTTP content type header field sent with the metafile). The Customizer-specific helper application will

then contact the Customizer to stream the video, which the Customizer retrieves from the Web server specified in the URL of the initial browser request. If the user relocates, the Customizer can be reloaded on a nearby CS to provide the best possible playback of the video stream.

IV. Performance

We briefly summarize some past results that demonstrate basic performance characteristics of our system, and then we describe a new experiment that highlights the advantages of using Customizers to improve the performance of video streaming.

A. Basic Customizer Overhead

The performance advantages derived from the ability to do remote customization can be negated if the underlying execution and communication mechanisms are slow. The use of Customizers introduces overhead because there are now two additional service points between Web client and Web server that operate in both directions. While we would like this overhead to be low in absolute terms, the primary goal is that it should be low relative to typical Web transaction times. Indeed, our measurements (presented in [29]) show that the overhead of our system when processing Web requests using a "null" Customizer (which simply passes them through) is less than 5ms on PCs with 933 MHz Pentium III processors running Solaris x86 release 2.8. This is small compared to Web transaction times that are typically 100-500ms (and these are for small requests; the overhead is completely negligible when transferring, say, images).

In the same study, we also measured the performance of an adaptive image filter Customizer, which highlights the end-to-end performance benefits of using a Customizer to provide adaptive image compression for a wireless client. The experiment simulated the common scenario in which a mobile client communicates with an access point over a wireless link, and the access point has a wired path to the rest of the Internet. We showed that using the CA to provide feedback regarding the available bandwidth at the client was effective for maintaining consistent transfer times as the wireless bandwidth changed.

B. Improving Video Streaming using an NFB Customizer

We now present an experiment that shows the benefits of using a Customizer that supports Network Flow Buffering (NFB) [30] to improve video retrieval and playback performance. The NFB Customizer is an implementation of a flow-regulating buffer, as described in Section III.D.

During a video session, the effective network bandwidth seen by the user may change as the result of changing network or server conditions. Reasons for this include changes in packet loss rate in, say, a wireless connection as the user roams, router congestion, or the server becoming overloaded and being forced to reduce the transmission rate. A NFB Customizer can be used to mask (from the client) dynamic changes in *relative* bandwidth between the WAN and LAN that can cause the effective bandwidth seen by the user to be lower than the video playback rate. The result is improved video playback performance by reducing the frequency and duration of interruptions to playback when the WAN and LAN bandwidths are highly variable relative to each other.

1. NFB Customizer Experiment

We performed the following experiment to validate the benefits of using a NFB Customizer. The experimental setup is shown in Figure 5. For the client, we used a notebook computer with a 500MHz Intel Pentium III processor running the RealOne™ Player on Windows 98. The server was a P3 933MHz PC running Windows 2000. The intermediate machine used to host the NFB Customizer (via a Customizer Server) was a Pentium II 450MHz PC running FreeBSD. To simulate a network with a given bandwidth, we used DummyNet in FreeBSD (IP Firewall kernel module), which supports the creation of pipes to control bandwidth, delay, and packet loss between two communication endpoints. Four pipes are used in total, one pair for symmetric bi-directional LAN control and another for symmetric bi-directional WAN control.

In this experiment, we show how a NFB Customizer provides improved playback that would otherwise be interrupted multiple times. In addition to the performance improvement, the NFB Customizer requires no more application-layer buffering than is required when streaming directly from server to client. In fact, under normal circumstances it is expected to use less. The video clip used for this experiment was 202 seconds (about 3.4 minutes) in length, with a bit rate of 38.5 KBps (308 Kbps).

DummyNet was used to create cyclical client/gateway "LAN" and gateway/server "WAN" bandwidths as shown in Figure 6. The figure shows the cycle repeating twice; the pattern actually continues repeating for the duration of the experiment. The averages for the WAN and LAN for each cycle are 41.75 KBps, which is above the average video playback rate of 38.5 KBps. Without the smoothing effect of the NFB Customizer, the effective bandwidth seen by the client is the minimum at any point in time of the bandwidth cycles, averaging 25 KBps, and we expect the video player to have to interrupt playback so that the network can catch up to the video.

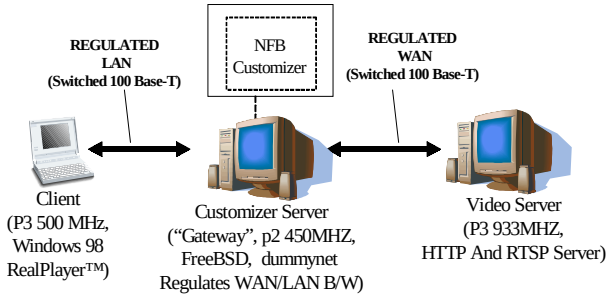

Fig. 5. Experimental setup.

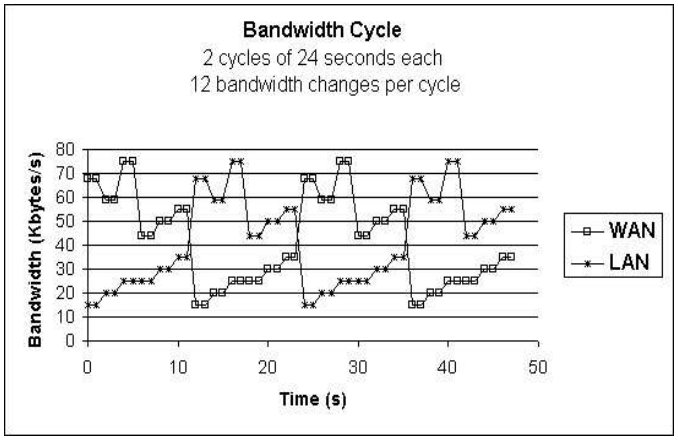

Fig. 6. Bandwidth for the video streaming experiment.

Figure 7 shows the video playback as amount of data played over time for direct HTTP, direct Real™, and NFB-enhanced HTTP, i.e., using a NFB Customizer. With NFB-enhanced HTTP, the player buffers for 17 seconds before playback begins. Once playback begins, the video plays smoothly and at full quality for its entirety without any additional buffering by the player. This is due to smoothing effect of the NFB Customizer, which allows a higher average bandwidth to be sustained to the client. For direct HTTP (i.e., without the NFB Customizer), the player buffers for 28 seconds before playback begins. During playback, the video plays at full quality, but the player interrupts the video three times to refill its buffer, for a total of 124 seconds of buffering after playback has started, and 152 seconds of total buffering, as compared to just 17 seconds with NFB-enhanced HTTP.

When the Real™ protocol is used, there is an initial 5 seconds of buffering before playback begins. The player quickly recognizes that bandwidth is inadequate and attempts to adapt accordingly. In doing so, it reduces the frame rate and picture quality, and ends up only retrieving 1.97MB (1966954 bytes), or about 25%, of the video. This results in a choppy, "slide show" resul ting in very poor-quality playback. Despite this adaptation, the video playback is still interrupted 7 times for a total of 165 seconds of additional buffering.

The extra buffering that occurs when there is no NFB Customizer is required because the player' s buffer is suffering from underflow, as the average bandwidth is lower than the video bit rate. Note that even if the player had perfect knowledge of the future, it would have to delay the start of the video by 152 seconds to ensure smooth playback under direct HTTP. Not only would this frustrate the user, it would also require a buffer size of nearly 6MB (152 x 38.5 KBps). Given that the player cannot anticipate network traffic bursts, or that it may not be desirable to delay the start of the video for so long and force the user to wait, or to reduce memory consumption, the player is forced to interrupt the video for two periods of half a minute or more to complete playback.

In Figure 8, we show the amount of buffering at the player for direct HTTP, Real™, and NFB-enhanced HTTP (in the latter case, the combined amount buffered at both the player and NFB Customizer is presented).

The large spikes in buffering for direct HTTP result from the player trying to manage its buffer when the bandwidth is bursty. The maximum buffer size for the player in this case is 1.1 MB (1103346 bytes). The buffering when using the Real™ protocol is also bursty. However, the drastic reduction in amount of video data played, at the cost of playback quality, results in a maximum client buffer size of 258 KB (257805 bytes). Finally, the effect of the NFB Customizer in NFB-enhanced HTTP can be seen by the significantly reduced burstiness of its (combined) buffering.

The combined buffering when NFB-enhanced HTTP is being used is further broken down into its two components, the client buffering and NFB Customizer buffering, in Figure 9. When the NFB Customizer buffer is peaking because the WAN bandwidth is higher than that of the LAN, the client buffer is draining because the LAN bandwidth is lower than the video rate. When the LAN bandwidth increases, the client

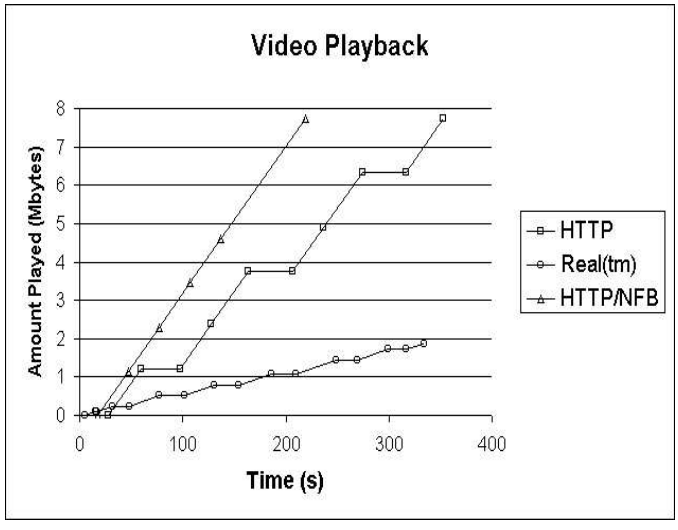

Fig. 7. Video playback for each of the three scenarios.

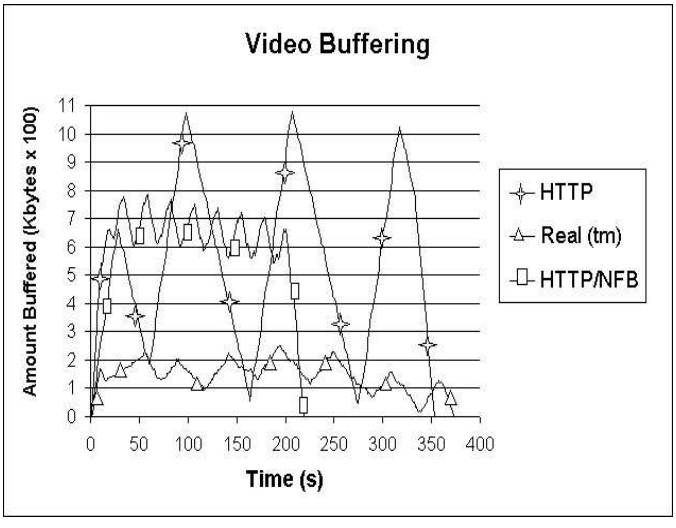

Fig. 8. Buffering during the video streaming experiment.

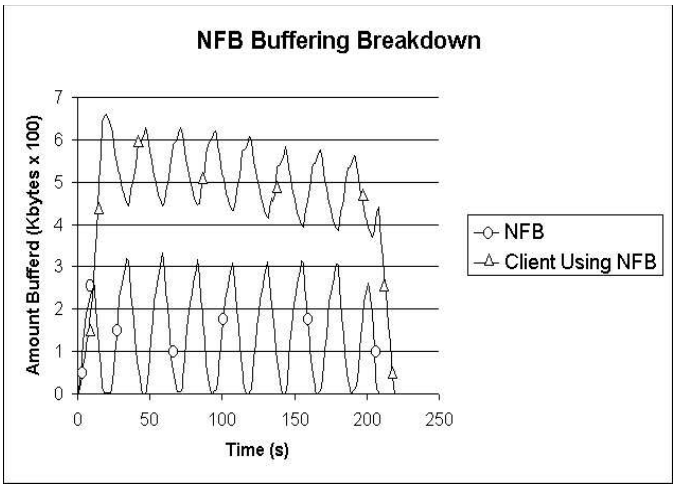

Fig. 9. Breakdown of combined buffering at client and NFB Customizer.

buffer will start to increase, but the NFB Customizer buffer begins to drain since the WAN is now the bottleneck. The maximum *combined* buffering (client + NFB Customizer) peaks at 785 KB (785057 bytes), less than the maximum buffer requirement at the client when direct HTTP is used. The maximum buffering *at the client* when the NFB Customizer is used is 660 KB (660514 bytes). This is due to the fact that without the NFB Customizer, the average effective bandwidth over the course of the playback is lower, so more buffering is required to compensate.

V. Related Work

Our work is premised on the idea that Web applications would greatly benefit from the remote customization capabilities of our system. In fact, there exists a large body of research results verifying the benefits of remote customization of Web content using proxies, mobile code, or some combination thereof [2, 7, 9, 27, 29]. Furthermore, there is a large body of research on the benefits of multimedia filtering, such as [9, 12, 21, 34]. We build on these results, as our distributed architecture allows these methods to be exploited more effectively.

A number of systems use a single remote proxy for customizing the Web, with communication initiated through the browser's proxy mechanism, including some in which the proxy is a personal server. This includes image and video filtering (sometimes called multimedia gateways), HTTP request modifications, HTML filtering, user interface improvements especially for small screens, remote caching, and support for disconnected operation and user-selected background retrieval [1, 5, 7, 8, 13, 22, 28]. Other systems have made use of the two-proxy (local and remote) concept for such customizations as filtering, prefetching, and intelligent cache management at the local proxy [22, 23]. In [11, 19] the server either on its own, or in cooperation with specialized proxies, works to customize content for clients or to improve the performance of prefetching and caching.

In [15] a pair of intermediaries is used to transparently provide fault tolerance, security, and timeliness in distributed object systems.

Research that is closest to ours combines the use of proxies or multimedia gateways with mobile code to support dynamic downloading of filters to a remote host [16, 25, 35]. There are also customization systems that do not use proxies *per se*, but rather use more general mobile code mechanisms to support remote processing at arbitrary hosts, typically at the servers themselves [14, 18, 20, 26, 27, 31, 32, 33].

A related issue is adaptability, where information is provided to the client application, typically from the operating system, to help it adapt to changes in resource availability and network connectivity [2, 4, 24]. Some of these systems include applications using an adaptable interface, including adaptable protocols.

The Internet Content Adaptation Protocol (ICAP) [17] is a solution developed by an industry coalition for distributing Internet content to edge servers. ICAP is server-centric in that distribution of functionality is controlled by content providers. This differs from the Web Stream Customizer Architecture, where the client controls the deployment of Customizers. Open Pluggable Edge Services (OPES) [3] is an IETF effort to standardize the tracing and control of proxies for content adaptation. Multiple edge services can be chained together, and callout servers can be used to offload computation. Their services can be server-centric or client-centric. Simple Object Access Protocol (SOAP) [6] is a simple XML-based protocol for exchange of information and RPC for Internet applications. None of the above approaches take advantage of mobile code to dynamically deploy a service, nor do they include the dynamic downloading of a local component on or near the client.

Our work differs from that of others in a number of ways. First, we optionally use personal servers (in addition to third-party servers) to store Customizers and their configurations, which facilitates Customizer reloading and provides dependable and trusted resources for the user's Customizers. Using third party and personal servers together enhances flexibility of the system, benefits performance, satisfies security needs of both the user and the Customizer authors, and provides a measure of fault tolerance.

Second, we have focused on a customization system designed specifically for the Web, allowing us to make a number of simplifying assumptions regarding the programming model, the user model, and the system design and implementation. We use a very restricted, and therefore more simplified, form of mobile code, rather than providing a generalized mobile code solution that, while more powerful, is less practical and is more complex in terms of usability and security. Other unique features of our system include the use of a CIS (Customizer Integration Server) that supports dynamic selection of multiple, simultaneously active, Customizers, and the use of CAs (Customizer Assistants) running on the CIS to support client-side processing and adaptability.

VI. Conclusions

We described the design of the Web Stream Customizer Architecture that provides the following capabilities:

- Dynamic placement of customization modules that can process and buffer data at intermediate points between a browser client and Web server
- The ability to adapt to network conditions because of its distributed control
- Support for user relocation by allowing efficient movement of customization modules

The architecture is tightly integrated with existing Web programming and communication models, and provides a customization framework that is ideal for wireless client devices. Reliance on a user-owned server, the PCMS, for execution of customization modules as well as their storage, increases practicality. The PCMS also plays an important role in providing support for Customizer relocation for execution on third-party servers that may be closer to the user. A soft-state execution model for Customizers simplifies the overall design, including Customizer relocation.

For future work, we plan to experiment with additional Customizer applications, especially those that rely on non-HTTP protocols for their basic communication.

REFERENCES

[1] E. Amir, S. McCanne, and R. Katz, "An Active Service framework and its application to real-time multimedia transcoding," *In Proceedings of ACM SIGCOMM*, pp. 178-189, Vancouver, Canada, August 1998.

[2] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, "System support for bandwidth management and content adaptation in Internet applications," *Proceedings of 4th Symposium on Operating Systems Design and Implementation*, pp. 213-226, San Diego, CA, USA, October 2000. USENIX Association.

[3] A. Barbir, R. Chen, M. Hofmann, H. Orman, and R. Penno, "An Architecture for Open Pluggable Edge Services (OPES)," *Network Working Group Internet Draft*. http://www.ietf.org/internet-drafts/draft-ietf-opes-architecture-04.txt

[4] V. Bharghavan and V. Gupta, "A Framework for Application Adaptation in Mobile Computing Environments, *Proceedings of IEEE Compsac '97*, Washington, D.C., USA, August 1997

[5] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul, "An active transcoding proxy to support mobile web access," *Proceedings of 17th IEEE Symposium on Reliable Distributed Systems*, West Lafayette, IN, USA, October 1998.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," http://www.w3.org/TR/SOAP/.

[7] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller, "Application-specific proxy servers as HTTP stream transducers," *Proceedings of 4th Intl. World Wide Web Conference*, pp. 539-548, Boston, MA, USA, December 1995.

[8] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd, "Power Browser: Efficient Web Browsing for PDAs," *Proceedings of CHI 2000*, The Hague, Netherlands, April 2000.

[9] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat, "Transcoding Characteristics of Web Images," *Proceedings of Multimedia Computing and Networking*, San Jose, California, USA, January 2001.

[10] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proceedings of ACM SIGCOMM*, pp. 14-26, Baltimore, Maryland, USA, August 1992.

[11] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters," *Proceedings of ACM SIGCOMM*, pp. 241-253, Vancouver, British Columbia, September 1998.

[12] S. Deshpande and W. Zeng, "Scalable Streaming of JPEG2000 Images using Hypertext Transfer Protocol," *Proceedings of ACM Multimedia 2001*, Ottowa, Ontario, Canada, October 2001.

[13] A. Fox, S. Gribble, Y. Chawathe and E. A. Brewer, "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives," *IEEE Personal Communications, Special Issue on Adaptation*, vol. 5, no. 4, August 1998.

[14] R. S. Gray, "Agent Tcl: A transportable agent system," *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, MD, USA, December 1995.

[15] J. He, M. A. Hiltunen, M. Rajagopalan, and R. D. Schlichting, "Providing QoS Customization in Distributed Object Systems", *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001*, pp. 351-372, Heidelberg, Germany, November 2001, LNCS 2218

[16] A. Hokimoto and T. Nakajima, "An Approach for Constructing Mobile Applications Using Service Proxies," *Proceedings of the 16th International Conference on Distributed Computing Systems*, Hong Kong, May 1996.

[17] Internet Content Adaptation Protocol. http://www.i-cap.org/home.html

[18] D. Johansen, R. van Renesse, and F. B. Schnieder, "Operating system support for mobile agents," *Proceedings of 5th IEEE Workshop on Hot Topics in Operating Systems*, Orca Island, WA, USA, May 1995.

[19] B. Krishnamurthy, C. E. Wills, "Improving Web performance by client characterization driven server adaptation," *Proceedings of the 11th International World Wide Web Conference*. Honolulu, Hawaii, USA, May 7-11, 2002.

[20] T. Kunz and J. P. Black, "An architecture for adaptive mobile applicatios," *Proceedings of 11th International Conference on Wireless Communications*, pp. 27-38, Calgary, Alberta, Canada, July 1999.

[21] J. Li, G. Chen, J. Xu, Y. Wang, H. Zhou, K. Yu, K. T. Ng, and H. Shum, "Bi-level Video: Video Communication at Very Low Bit Rates", *Proceedings of ACM Multimedia 2001*, Ottowa, Ontario, Canada, October 2001.

[22] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, and K. Raatikainen, "Optimizing World-Wide Web for Weakly-Connected Mobile Workstations: An Indirect Approach," *Proceedings of 2nd International Workshop on Services in Distributed and Networked Environments (SDNE)*, pp. 132--139, Whistler, Canada, June 1995.

[23] T. S. Loon and V. Bharghavan, "Alleviating the latency and bandwidth problems in www browsing," *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, Monterey, CA, USA December 1997. URL: http:/timely.crhc.uiuc.edu/.

[24] B. Noble, "System support for mobile, adaptive applications," *IEEE Personal Computing Systems*, vol. 7, no. 1, pp. 44-49, February 2000.

[25] W. T. Ooi, and R. van Renesse, "Distributing Media Transformation Over Multiple Media Gateways," *In Proceedings of ACM Multimedia 2001*, Ottowa, Ontario, Canada, October 2001.

[26] H. Peine and T. Stolpmann, "The Architecture of the Ara Platform for Mobile Agents," *Rothermel K., Popescu-Zeletin R. (Eds.), Mobile Agents, Proceedings of MA '97*, pp. 50-61, Springer Verlag, Berlin, Germany, April 7-8, 1997, LNCS 1219.

[27] S. Perret and A. Duda, "Implementation of MAP: A system for mobile assistant programming," *Proceedings of IEEE International Conference on Parallel and Distributed Systems*, Tokyo, Japan, June 1996.

[28] H. Rao, Y. Chen, M. Chen, J. Chang, "A Proxy-Based Personal Portal", *Proceedings of the WebNet99 Conference*. Hawaii, USA October, 1999.

[29] J. Steinberg and J. Pasquale, "A Web Middleware Architecture for Dynamic Customization of Content for Wireless Clients," *Proceedings of the 11th International World Wide Web Conference*. Honolulu, Hawaii, USA, May 7-11, 2002.

[30] J. Steinberg and J. Pasquale, "Using Network Flow Buffering to Improve the Performance of Video over HTTP," *University of California,, San Diego Technical Report number CS2004-0776*. January 14, 2004.

[31] M. Straßer, J. Baumann, and F. Hohl, "Mole - A Java Based Mobile Agent System," *Proceedings of the ECOOP'96 workshop on Mobile Object Systems*, Linz, Austria, July 1996.

[32] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal, "Active Names: Flexible Location and Transport of Wide-Area Resources," *Proceedings of the Second Usenix Symposium on Internet Technologies and Systems*, Boulder, CO, USA, October 1999.

[33] Y. Villate, D. Gil, A. Goni, and A. Illarramendi, "Mobile agents for providing mobile computers with data services," *Proceedings of the*

Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98), Newark, DE, USA, October 1998.

[34] N. Yeadon, A. Mauthe, D. Hutchison, and F. Garcia., "QoS filters: Addressing the heterogeneity gap," In *Proceedings of IDMS ' 96 European Workshop on Interactive Distributed Multimedia Systems and Services*, Berlin, Germany, March 1996.

[35] Zenel and D. Duchamp, "A general purpose proxy filtering mechanism applied to the mobile environment," *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 248-259, Budapest, Hungary, September 1