Launch Support Video Site

Zachary L. O'Farrell

Kennedy Space Center

Computer Science

KSC-FO Summer Session

Date: 7-29-2013

Launch Support Video Site

Zachary L. O'Farrell[1]
Colorado State University, Fort Collins, Colorado, 80523

The goal of this project is to create a website that displays video, countdown clock, and event times to customers during launches, without needing to be connected to the internal operations network. The requirements of this project are to also minimize the delay in the clock and events to be less than two seconds. The two parts of this are the webpage, which will display the data and videos to the user, and a server to send clock and event data to the webpage. The webpage is written in HTML with CSS and JavaScript. The JavaScript is responsible for connecting to the server, receiving new clock data, and updating the webpage. JavaScript is used for this because it can send custom HTTP requests from the webpage, and provides the ability to update parts of the webpage without having to refresh the entire page. The server application will act as a relay between the operations network, and the open internet. On the operations network side, the application receives multicast packets that contain countdown clock and events data. It will then parse the data into current countdown times and events, and create a packet with that information that can be sent to webpages. The other part will accept HTTP requests from the webpage, and respond to them with current data. The server is written in C# with some C++ files used to define the structure of data packets. The videos for the webpage will be shown in an embedded player from UStream.

Nomenclature

KNET	=	KSC private network
OPSNET	=	Telemetry and Communications internal network
DOM	=	Document Object Model
IP	=	Internet Protocol
XML	=	Extendable Markup Language
LSP	=	Launch Services Program
KSC	=	Kennedy Space Center
URL	=	Uniform Resource Locator

I. Introduction

The Telemetry and Communications summer 2013 intern project is to create a webpage to be used for Launch Services Program (LSP) to display streaming video and the countdown clock. The final product will help support mission related and day to day activities. To accomplish this, the video will be embedded into the webpage using the Ustream embedded player. The requirement for the clock data is that it be accurate, with a maximum delay of two seconds.

II. Related Software and Programming Languages

A. Ustream.tv

Ustream.tv is a website that allows video to be streamed live through their website, from multiple platforms. For this project, the video was broadcast from an encoder to Ustream. An encoder is a device which converts a video signal to an IP stream. Once Ustream receives the video, Ustream disseminates it using their content distribution network.

---

[1] KSC-FO Intern, LSP Telemetry and Communications, Kennedy Space Center, Colorado State University.

B. Microsoft Visual Studios

The development environment for this project was Microsoft Visual Studios, and was used for both the webpage and the clock relay app. Visual Studios is capable of supporting all of the programming languages with auto-complete and real-time error checking.

III. Development

This project has two parts that work together to satisfy the requirements, a webpage and a data relay application. The webpage's two functions are to display Ustream video, and countdown clock. The data relay application receives multicast packets that contain the data for the countdown clock, and sends the information to the webpage.

A. Webpage Video

Displaying the video stream from Ustream was accomplished by embedding the stream in an iframe. A HTML frame is an element that splits the browser into different segments. The segment can be updated separately from the rest of the page, which allows the video to be continuously streamed without the rest of the page being affected. An inline frame, or iframe, is a frame that does not have to be in a frameset, and can instead be inserted anywhere in the HTML code. The advantage of using an iframe for the Ustream video is that the code for the embedded player is handled by Ustream. To play the video, the iframe's source points to an ustream.tv URL which is specific to the video. When the page loads, the iframe uses the URL to get the embedded player from Ustream and play the video. The biggest reason for using the Ustream player and the iframe is that Ustream can add features to their player, which will automatically show up on the webpage, with no additional work to the webpage. An example include an interactive menu that allows for playing, pausing, seeing how many people are watching the same video. If an iframe is not used, these features would have to be implemented in the webpage which would add time to the development and testing of the webpage.

Figure 1: Video iframe is outlined in red

The number of videos that would be streaming to the webpage is currently three, but that number is likely to change in the future. Because the number can be changed, hardcoding a set number of videos into the webpage is impractical and would require the page to be re-written each time a video was added. The final layout has one video, and a menu for selecting which video to watch. Changing the video is done using a JavaScript function that manipulates the DOM elements for the video. Each element in the video menu has the name and URL of the stream. When someone selects an option, the JavaScript function is called and the video and its title are changed. The function is able to get the menu by using its ID, and then from that menu, the current selection is retrieved, and the video data is updated.

B. Webpage Countdown Clock

The most important part of this project is to have accurate clock data displayed on the webpage. Clock data consists of time about liftoff, such as L-Time and expected liftoff, and what events are coming up and when they will occur. The accuracy requirement is that the time displayed is no more than two seconds different than the actual time and ideally the delay would be less than one second.

L-Time 000:00:00:00	Liftoff 000:00:00:00	T-Time 000:00:00:00
Window Remaining 000:00:00:00	Window Close 000:00:00:00	Built-in Hold 000:00:00:00

Figure 2: Countdown clock with the different time fields

The first approach at satisfying this requirement was to stream a video of the clock to Ustream, and make it available as one of the video options. This is the easiest way to complete the project, because it uses one solution to

meet both the video and the clock requirements, and it alleviates the need for a separate program to relay clock data to the web page. However, this solution does not meet the accuracy requirements for the data. During testing, the least delay that was seen was approximately six seconds. This was when the video was streamed to a desktop with a wired connection to the internet. On tablets and phones that were connected to Wi-Fi, the delay was at least 30 seconds, and often longer. Because of these violations, a separate application had to be used to stream clock data to the webpage.

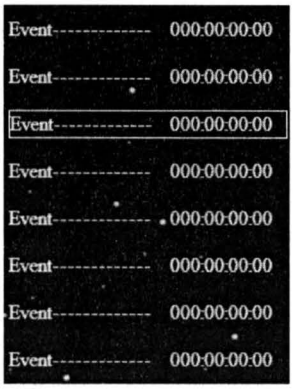

Figure 3: Event box, the current or next event has a white box around it

To get the data from the relay application, the webpage uses a JavaScript function that sends a HTTP request to the program, and the data is returned in the HTTP response. The function is run on a timer so that once every second; the page requests a new data packet, which contains the latest clock data. When the packet returned, the JavaScript function parses the data, and updates the webpage. The format of the data is "key|||value" with each key being the id of an HTML DOM element, and value being the current value of that element. Each key-value pair is separated from others by a ';'. One second after the previous request was processed, a new HTTP request is sent, and the data is updated again. This cycle continues as long as the webpage is open.

One problem with using HTTP requests is that browsers cache requests based on the IP address and port number. When the browser gets a request that has been cached, it returns the same response that was returned the time before. This causes the times to only be updated when the webpage starts the first time, and the browser returns the cached value for all subsequent requests. To fix this problem, each HTTP request has a random number added to the end of the URL as a parameter. This prevents the browser from thinking that two requests are the same and returns a unique response each time. This method for getting clock data to the webpage has a delay of less than two seconds, even when the program has to handle a large number of requests.

C. Webpage Layout

The number of videos, size of the page, and the ability to easily read the clock determined how the page was laid out. Each of these helped create the best and easiest to use layout for the user.

The biggest variable in the page is the number of videos that could be streamed at one time. At the start of the project, only three videos were used, so all three videos could be on the page at one time. A problem with this was that each video was small, and hard to see. To overcome this, the layout was changed to feature one big video, and several smaller ones. Any of the smaller ones could be selected to switch it with the biggest video. The big one was easy to see, and had the main focus of the page, while the other videos were just big enough to be able to tell what was going on. These smaller videos acted more like a thumbnail, and were useful for determining when to switch to a different stream and what was on it. The problem with this layout is that when more videos are added, part of the page will have to be re-written to accommodate more videos. Each video added will also decrease the size of the other videos, making them harder to see. To come up with a solution to handle an unknown number of videos, the small videos were replaced with a pull down menu. The menu contains each video stream and name, and when the user selects a stream, it replaces the currently playing video. Adding videos to the menu is very simple, and only requires adding one line of code to the HTML file. In future versions of the website, multiple pages can be used and each one can have its own layout. This would allow a user to select the best configuration for his or her needs. An example of this could be to have a page with all of the videos on it, filling the entire page. This would hide all clock information, but give the user access to every stream at once.

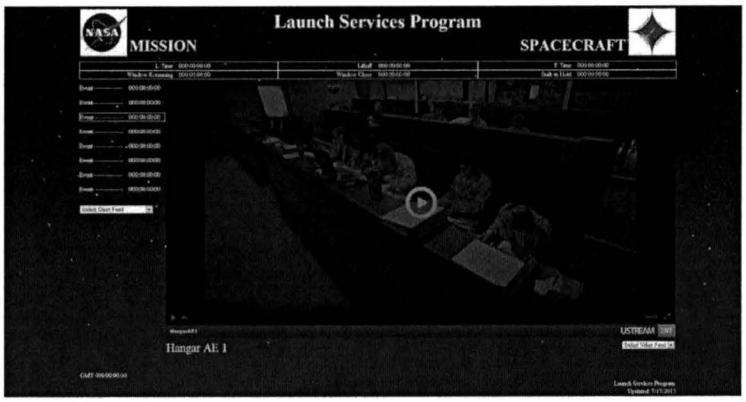

Figure 4: Webpage fitting in a large browser window

Another driving factor for the page layout was the size of the page. This page needed to fill as much of the browser window as possible, without using

scrollbars. This allows the video to be as big as possible without being cut off by the browser. The video also needs to grow and shrink as the browser expands. Figures 4 and 5 show the page changes when the browser size is changed. Figure 4 shows the page when the browser fills the entire screen, and figure 5 shows the webpage when the browser is about half the size. The biggest difference between the two is that the size of the video in the bottom picture is smaller, to fit the size of the browser. Preventing the page from having to scroll keeps the video from being cut-off when the page gets smaller.

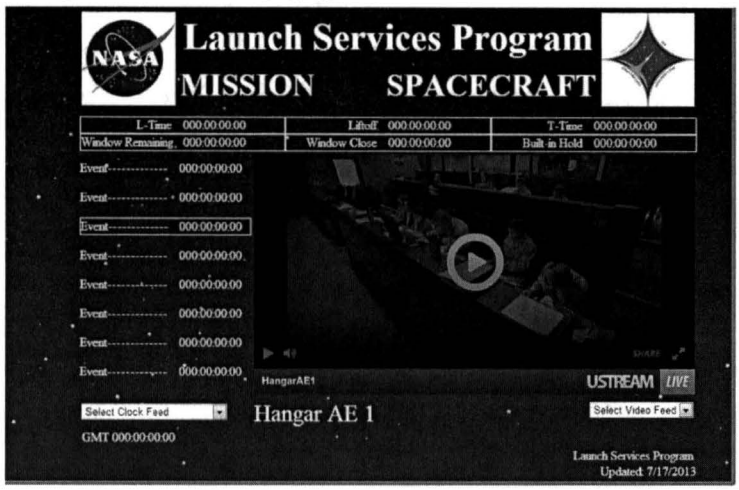

Figure 5: Webpage fitting in a large browser window. Compare the height of the video to the height of the events, which does not resize based on the browser.

The final major factor in creating the layout was how easy it is to read the clock data. The clock is the most critical part of the webpage, and needs to be the main focus. The clock data has two parts, the events and countdown clock. The page has the countdown clock data at the top of the page, so it is the first thing the user looks at when getting to the page. The font style was changed so that the text is bigger and the color of it is white, so that it pops out against the page better. Each event in the event section has the name of the event, and the L-Time that is it supposed to occur. The events section shows two previous events, the current or next event that will occur, and the five events after that. The current event is the most important and needs to be easily identified. To do this, a box is drawn around event. Without this, the only way to tell which events have already passed is to compare the current L-Time against the event L-Time time. Using the box makes this process much quicker and easier for the user.

D. Relay Application Receiving Data

The application that relays data from the Operations Network (OPSNET) to the webpage is broken into two pieces. One part receives and parses data received from the OPSNET and the other sends the information to the webpage.

The clock data is sent out on the OPSNET in multicast packets. Clock data is separated into three packets, one for countdown clock, and two for events, and they are all sent in order each time a new packet is generated. The relay program subscribes to that multicast group and receives all packets that are output by the clock. The packets are read in as byte array, which is then parsed and the information is stored in an object that can be sent to the webpage.

Each of the three packets have a different structure, so each one needs its own parsing method, but all three use an integer to identify its type as the first four bytes in the packet. This integer is read first, and that determines how to parse the byte array.

```
Select Clock Packet Type
    READ data packet into Clock
    CONVERT first 4 bytes of Clock into integer Type
    IF Type = 1 THEN
        Call parseCountdown with Clock
    ELSE IF Type = 2 THEN
        Call parseEvents with Clock
    ELSE
        Ignore packet
    ENDIF
```

After the data packet has been parsed, it is stored in a web packet object. This packet contains a string with key-value pairs separated by a ';', as described in section B. The data packets received contain more information than is needed for the webpage, so some has to be discarded. There are only seven countdown values that are used, but

many more are included in the packet. Also, the events packets can contain up to 20 events each, for a total of 40. The webpage only cares about eight events total, and they are always contained in the first event packet, with a packet type of two. Because of this, the second packet, with a packet type of three, is always ignored by the program. Being able to not use some of the data in the packets helps the performance, because it decreases the amount of processing that has to be done on the data, and makes the packet sent to the webpage smaller. The relay application receives all of the clock data in three packets, but it is able to send out just one packet. After the two packets that the program is interested in are read and the packet is sent, the packet is reset and the relay application waits for the next countdown information packet. This loop is repeated for as long as the program is running.

E. Relay Application Communicating with the Webpage

The final part of this project, and the relay application, is to be able to send the clock data to the webpage. This part takes in the data to be sent as an array of bytes, which can be sent over the network to the webpage. Each webpage makes a request whenever it wants new data. The program uses a HTTPListener object which runs on a thread and waits for HTTP requests to be received. Once a request is received, it is passed to a new thread which handles responding to it. The new thread is used so that the HTTPListener can receive more requests. If threads were not used, each request would have to be completely responded to before another request could be received, which would slow down the execution of the program. Creating a thread is an expensive operation, so instead of spawning a new thread for each request, a thread pool is used. A thread pool uses a pre-set number of threads that are created at the start of the program, and a queue of tasks to be completed. Whenever a new HTTP request comes in, it is added to the queue of work waiting to be done. When there are requests in the queue, the next free thread gets the request at the front of the queue and responds to it. That thread is then freed to complete the next task at the head of the queue.

One problem with using threads is keeping the program thread safe. Thread safety has to be taken into considerations when more than one thread has access to a variable that can be modified. For this application, the mutable data is the web packet that is sent from the relay application to the webpage. The data is being sent out in response to the HTTP requests, and at the same time that data is being updated in response to new data being received. To keep the data in a consistent state, a readers-writer lock was used on the webpage packet. The readers-writer lock allows for multiple threads to access the data as a reader, but only one thread to write to it. When a thread wants access to the data, as a reader, the reader checks the state of the writer lock. If it is taken or a thread is waiting to take it, the reader waits. If no one has or wants the writer lock, the reader is able to get that data. Once the reader is through with the data, it releases its lock. There is no limit to the number of readers who can have a reader lock at the same time. The writer lock, however, can only be accessed by one thread at a time. To acquire the writer lock, the writer checks to see if anyone has the writer lock, or is waiting for it, and if not it takes the lock. The writer also has to worry about if there is a reader accessing the data. After checking for other writers, the writer checks the number of readers currently accessing the data. If the number is greater than zero, the writer blocks any other readers from getting the data, and waits for all current readers to exit. Once all other readers have exited, the writer is able to modify the data. After it is finished, the writer releases its lock and readers are able to access the new data. Other methods of locking data will not distinguish between readers and writers, and only one thread can access the data at one time, no matter what it will do. The performance boost that this method has is that the only time you have a single thread accessing the data is when you have to modify it, and the rest of the time any number of threads can access the data.

Using the readers-writer lock increases the parallelization of the program, but it creates a bottle neck when something tries to write to the data. This becomes more of a problem as the number of writes increases. For this program, clock data is received several times per second, so several times a second the data is inaccessible for readers. Writing to the data is also a slower operation, because the data has to be converted from a byte array into individual values and then written to the webpage data packet. To help decrease the time that the data is locked, the program has two copies of the web packet. These packets contain the same variables, but one is used for writing. As data comes in, this packet is updated, and the other one is still available for reading. Once the new packet is completely written, it replaces the reading packet, and the data in it is cleared. Replacing the reading packet is faster than updating it, because it just involves copying the data from one variable to the other, no other processing has to be done.

```
Create Web Packet
    WHILE receiving data
        CONVERT clock data INTO values
        FOR EACH value IN values
```

```
        STORE value IN writerWebPacket
    END FOR
    OBTAIN writerLock FOR readerWebPacket
        SWAP writerWebPacket FOR readerWebPacket
    RELEASE writerLock FOR readerWebPacket
    CLEAR writerWebPacket
END WHILE
```

IV. Conclusion

. At the conclusion of the 2013 summer internship session, although the webpage is not in use, all of the programming is completed so that it can quickly be deployed. The page successfully streams video from Ustream and resize the video to effectively use the space available on the page. It also streams clock data from the relay application to the webpage with minimal delay. The webpage will be deployed once it is approved by LSP management and thoroughly tested.

Acknowledgments

Zachary O'Farrell would like to thank the following people for help and input into the project:

Reed Divertie
Jarel Lawrence
Dan McNerney
Nate Wood
Eric Anderson