Aspera FASPstream for near-live and live video streaming

An Aspera Labs Whitepaper

April 2015

WHITE PAPER

aspera
an IBM® company

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

TABLE OF CONTENTS

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

INTRODUCTION

Live and near-live streaming of broadcast quality video content (10 - 40 Mbps) over IP networks with small start-up delays and glitch free experiences have traditionally required expensive and specially provisioned infrastructure. Traditional distribution systems use live satellite feeds from the streaming source or dedicated terrestrial networks with heavy quality of service to ensure low latency and low packet loss rates so as to not degrade the play out quality. In recent years, advances in broadcast quality video compression have also made it possible to distribute lower bit rate versions through web streaming over consumer broadband bandwidths (1-10 Mbps), but even at these bit rates, providers have relied heavily on global content distribution networks (CDNs) to stream the video content from the network edge to the consumer. These CDNs allow the user to take advantage of low round-trip latency and relatively low packet loss rates or a better quality user experience. Note that lower bit rate transcoding does not eliminate the need to ingest a high quality stream to transcoders, which may be remote from the source.

At both ends of the spectrum — broadcast quality ingest and remote play out as well as consumer web streaming — the media enterprise and, indirectly, the consumer pay a heavy premium in infrastructure costs to minimize the network factors that degrade the play out experience, such as network round-trip time and packet loss. This cost is burdensome in any media application or service, but is especially impractical in live and second screen experiences for events that occur only once. One time events such as sport events, movie premieres, concerts, operas, etc. cannot as easily justify the investment associated in erecting dedicated infrastructure for direct distribution, or amortize the CDN costs over long periods of viewing. Additionally, there exist practical constraints that make it difficult to employ CDNs for direct distribution in many second screen applications; media need flow through distant cloud services where scale out properties of the cloud computing platform are necessary for concurrent transcoding of the live stream for several formats. Thus, more often than not, content providers are left to over-provision infrastructure for such live events as a precaution, and pay higher costs.

The need to ingest and distribute live media over low-cost, wide area IP networks, and with the option to go through distant cloud-based transcoding platforms has created a significant technology void. This is not just an opportunity for small incremental advantage solvable through adaptive bit rate "down sampling", or clever buffering schema – instead, this calls for a fundamental solution.

Traditional TCP-based transport approaches such as adaptive bit rate streaming over HTTP have a significant bottleneck in throughput over commodity Internet WANs. The best case transfer rates for live HTTP streams over a commodity Internet WAN path between South America and Europe (200 milliseconds round-trip time) is 2 Mbps, and with worst case internet packet loss rates (≥ 2%), falls to < 1 Mbps. A 10Mbps live ingest stream transported with TCP simply isn't possible over these distances. And for higher video bandwidths or more distant delivery, the gap between realizable throughput and required play-out rate widens.

Aspera FASP transport technology is a patented bulk data transport widely utilized in digital media for achieving highly efficient, high-speed bulk media transfer over IP networks, with efficiency independent of distance and quality (round-trip latency and packet loss). However, until this point, the FASP architecture had no suitable application interface for transporting live data. In contrast with bulk file-structured data (e.g. VoD), live stream data need be delivered to the play out application in the same order it was passed to the transport tier. This ordered delivery constraint required Aspera to innovate a new byte streamlining capability in its transport platform on top of the FASP datagram delivery protocol. The resulting protocol— "FASPstream" — is a fully reliable bulk data streaming protocol that delivers data and video streams over Internet WANs including minimal buffering or glitches, and with negligible start-up delay. In this paper we describe the underlying protocol design and statistical model that predicts the FASPstream performance and we demonstrate through real world measurements the resulting quality in live video delivery that creates radically new possibilities for live streaming video. Media companies can achieve long distance ingest, remote play out, and even distribution of live video to play out systems running FASPstream without the assistance of CDN edge technology, "game changing" capabilities that could ultimately revolutionize the transport of live video.

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

Aspera has productized the FASPstream technology as embeddable SDK bindings available for C/C++, .NET, and Java for use in 3rd party applications, and as a new capability in its core 'ascp' transfer binary for use cases interoperating live streams with remote file storage and vice versa. The first production use case delivered live video during the World Cup in a pioneering second screen system that captured 14,000 hours of live video from multiple camera angles, ingested this video in real time using Aspera FASP from Brazil to an AWS cloud storage for live transcoding, ultimately yielding approximately 3 million minutes of transcoded content served by broadcasters, and ultimately to consumers.

We begin with a tour of the mathematical model that underlies the protocol, and predicts its efficiency. Then we generate live stream data on networks similar to those described above—200 ms delay, 2% packet loss— and three representative bit rates—6Mbps, 10Mbps, 40Mbps—to compare to our expectations. These actual transfer data sets confirm both our model, and the transfer protocol as bona fide solutions to the streaming video challenge. We end with some simple visualizations to expose the hard numbers that FASP- stream achieves in terms of skips per second of video playback.

THE FASPSTREAM MODEL

Consider a live video stream being "played out" over a wide area IP network. Viewers react to two important measures of "quality" — how long the stream takes to start playing (the start up delay), and whether the stream plays smoothly or "glitches"/"buffers" waiting for the expected next data to arrive (the "glitch" probability). Our formal evaluation of the Faspstream model in this section considers both facets of quality.

START-UP DELAY

Let's start with quantifying the start-up delay. If all of the packets that comprise the video stream were to always arrive at the play out location on the initial attempt, to determine the delay before play out begins, we would need only ask, what is the One-Way-Transfer time for the network? Simply knowing that time would determine how long the first packet in the stream would take to arrive, and all of the remaining packets would also arrive precisely in time for the player. However, IP networks are imperfect and merely "best effort" by design: packets are lost or delayed in transit and there is no guarantee that any packet will arrive at its destination on the first attempt or at any particular time!

Assuming a typical full duplex intercontinental IP network with a 200 millisecond round-trip time (RTT) and one way propagation delay (OWD) of 100 milliseconds and loss probability of 2% in each direction, for a reliable delivery protocol like HTTP over TCP, the transmission rate has been shown to collapse to an effective rate of less than 1 Mbps, due to the congestion windowing protocol of TCP that reduces the sending rate aggressively in response to packet loss. Thus "live" transmission and play out of data stream rates greater than 1 Mbps using TCP based protocols is impossible at such network distances because start-up delays to compensate are unbearably long.

The Aspera FASP protocol, however, has a reliability algorithm fully decoupled from its congestion avoidance algorithm and independent of round-trip delay. If data is dropped, the FASP receiver requests retransmission of dropped packets by the sender, and does not slow down its the sending rate; packets are retransmitted along with new data at the effective bandwidth rate of the channel. So, if the FASP design principals hold for in-order stream data, we can predict precisely how long we need to wait for a video stream to begin.

For example, we conclude that for a live stream over a WAN with 2% packet loss, on average, only one in every 50 data blocks will be dropped, and for 50 blocks to arrive, we need wait only the One-Way-Delay time, plus the time to retransmit a block, 1 round trip time (RTT). While close to accurate already, this doesn't entirely capture the story. Let's scale up a bit.

Consider 2500 blocks to be sent. 2% of these blocks are expected to drop, i.e. 50. For each of these 50 blocks, we have two possibilities:

- The message requesting block retransmission sent by the receiver to the sender could be dropped in transit. This may occur with probability 2%
- The retransmitted block might be dropped on its second trip. This may occur with probability 2%.

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

One can readily compute that we expect 1 of our 50 blocks to be dropped a second time, and 1 of our retransmission requests to be dropped. How would this in turn affect the stream delivery time? Of all blocks, we would expect 2450 to arrive on time, that is, after the One-Way-Delay time. We expect 48 to arrive after the retransmission request. This is 1 RTT plus 1 OWD. For the two remaining blocks (the double dropped blocks), their arrival delay will suffer another RTT; arrival will occur in 2 RTT + 1 OWD.

Finally, we consider the special case that if the final block's retransmission request is dropped, the FASP protocol will detect this case and one RTT later, send another retransmission request. Then the block is retransmitted successfully (with 98% likelihood). This entire process takes 1 RTT + 1 RTT + 1 OWD. Thus, we have two blocks arriving after 2 RTT + 1 OWD. To give a sense of this in human time, RTT in our model network is 200ms, and OWD is 100ms. Thus, in 2500 blocks, all but two arrive within three-tenths of a second, and the last two, in three-fifths of a second. This means that we can ensure a worst-case start up delay of only three-fifths of a second, assuming FASPstream can continue to deliver data at the play out rate and with no data pre-buffered in reserve.

Lest these predictions of block numbers seem sleight-of-hand, consider the actual calculated number of blocks for some sample tests. For 6Mbps data streams, 547 blocks are sent per second; for 10Mbps streams, 911 blocks are sent per second, and for 40Mbps streams, 3641 blocks are sent per second. Compare these numbers with the sample numbers described above, and you'll get an immediate sense on the performance we are expecting. Later we will show some real data to compare, but for now, these are concrete numbers to keep our feet on the ground.

"GLITCH" PROBABILITY AND THE FULL MATHEMATICAL MODEL

The more interesting question in characterizing the FASPstream performance is how to know the probability that the live stream will "glitch" during playback because the next expected data is not available to the player, and how much additional start up delay is needed to ensure enough data is stored in reserve to prevent this given the network conditions. To formalize this, we set out to compute a probability model for "skipping" a frame. Specifically, we want to know the probability that a skipped block will not be received in time for its expected playback.

First we need consider how the FASP datagram size equates to playback time. Video is delivered in a number of frames per second (the 'framerate') and has a number of bytes per frame. For one second of live video, the number of data bytes the FASPstream will send is framerate*framesize. In other words, the FASPstream transmission rate in bits per second is video framerate *framesize *8. We assume also that the FASPstream transport reads the video stream from the video source in "readchunks" of a known size, and delivers these chunks to the receiving player application in order (while individual blocks within the chunks may arrive out of order). It turns out that the size of the chunk has no affect on the formal model.

To determine the probability of a "glitch" in the live transmission, we need to first compute the probability $\mathcal{P}$ of waiting a number M RTTs before the next needed chunk arrives and then normalize M for the playback rate of the video.

Note: *The absolute baseline for any block to travel the network is 1 OWD. For this reason, we normalize by this OWD and don't include it in all of our computations. When we say "wait M RTTs", we always mean "wait M Rtts after the first OWD".*

The probability model is an iterated probability of a single block failing. For streaming files, order matters, and thus the delay of playback is the result of a packet being dropped multiple times.

The probability of waiting one RTT for one block is P, i.e. the probability that one block is dropped. The probability of waiting two RTTs for one block is P^2, or, the probability of that block being dropped, and then being dropped again. This may be iterated for M RTTs.

Hence, the probability of a block not being recieved in M RTTs is P^{M+1}. The index change is to reflect precisely how many RTTs are needed to *recieve* the block, i.e. the block arrives *within* M RTTs.

From this we can see that the probability that a block *will* be received in M Rtos is $1 - P^{M+1}$. Now given two blocks, the probability that we receive the first block within M RTTs, AND we receive the second block within M RTTs is the product of their individual probabilities. This is because each block transmission is an independent event. Whence:

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

$$\mathcal{P}(M,2) = (1 - P^{M+1})(1 - P^{M+1}) = (1 - P^{M+1})^2. \quad (1)$$

Now, assume *N* is the number of blocks in a single readchunk. By this we mean *N* is the readchunk size divided by the block size. This unit represents the minimal number of blocks to begin playing.

Now, the probability of receiving *N* blocks in *greater than M* RTTs is

$$\mathcal{P}(M,N) = 1 - (1-P^{M+1})^N \quad (2)$$

This is a result of the previous fact: that receiving all *N* packets in *M* RTTs is the product of their individual probabilities.

SOME EXPECTATION VALUES

To test our theory we ran a number of experiments to create a dataset consisting of 100 transfers of 30 seconds of video, using two host computers connected over an IP network (10 Gbps Ethernet end-to-end) via a network emulation device configured with 200ms round trip delay and 2% packet loss. We created data streams at 40, 10, and 6 Mbps via netcat to serve as representative video bit rate examples. We assumed that the available bandwidth capacity of the network was greater than the data stream rate and configured the emulated channel bandwidth to 50, 15, and 10 Mbps respectively. Finally, we captured the arrival timestamps of all packets at the FASPstream receiver host and recorded the arrival rate for original and retransmitted packets by block number and by readchunk for our analysis.

For our three cases of 40Mbps, 10Mbps, and 6Mbps we estimate 3641, 911, and 547 blocks per readchunk, where we assume the readchunks are 1 second long, assuming a block size of 1464 bytes. We assume that all FASPstream applications can have a 1 second start-up delay and thus pre-buffer one chunk of one second duration, which translates to 5 RTTs on a 200ms network. We assume a packetloss probability of 2%, and calculate the probability that any 1 chunk will be delayed more than the 1 second of buffer and cause a "glitch" as follows:

$$\mathcal{P}(5, 547) = 1 - (1 - .02^5)^{547} = 0.00000175 \quad (3)$$

$$\mathcal{P}(5, 911) = 1 - (1 - .02^5)^{911} = 0.00000292 \quad (4)$$

$$\mathcal{P}(5, 3641) = 1 - (1 - .02^5)^{3641} = 0.00001165 \quad (5)$$

After running 100 tests of 30 seconds of live data(30 chunks) the probability of a "skipped" chunk is as follows:

$$0.00000175 \times 30 \times 100 = 0.00525 \quad (6)$$

$$0.00000292 \times 30 \times 100 = 0.00875 \quad (7)$$

$$0.00001165 \times 30 \times 100 = 0.03495 \quad (8)$$

This data tells us that with a one second buffer, at 6 and 10 mbps, we should skip less than 1 percent of the time, and even at 40 mbps, we should skip less than 4% of the time.

Video bit rate	Time before expected skip
6Mbps	6.6 days
10Mbps	3.96 days
40Mbps	0.99 days

Our tests of the FASPstream experimentally confirm these bold claims. We see exactly as predicted, zero observable skips, and the majority of frames arrive ahead of time, suggesting the buffer time can be reduced under a second (on the order of 0.5 seconds).

Aspera FASPstream for near-live and live video streaming

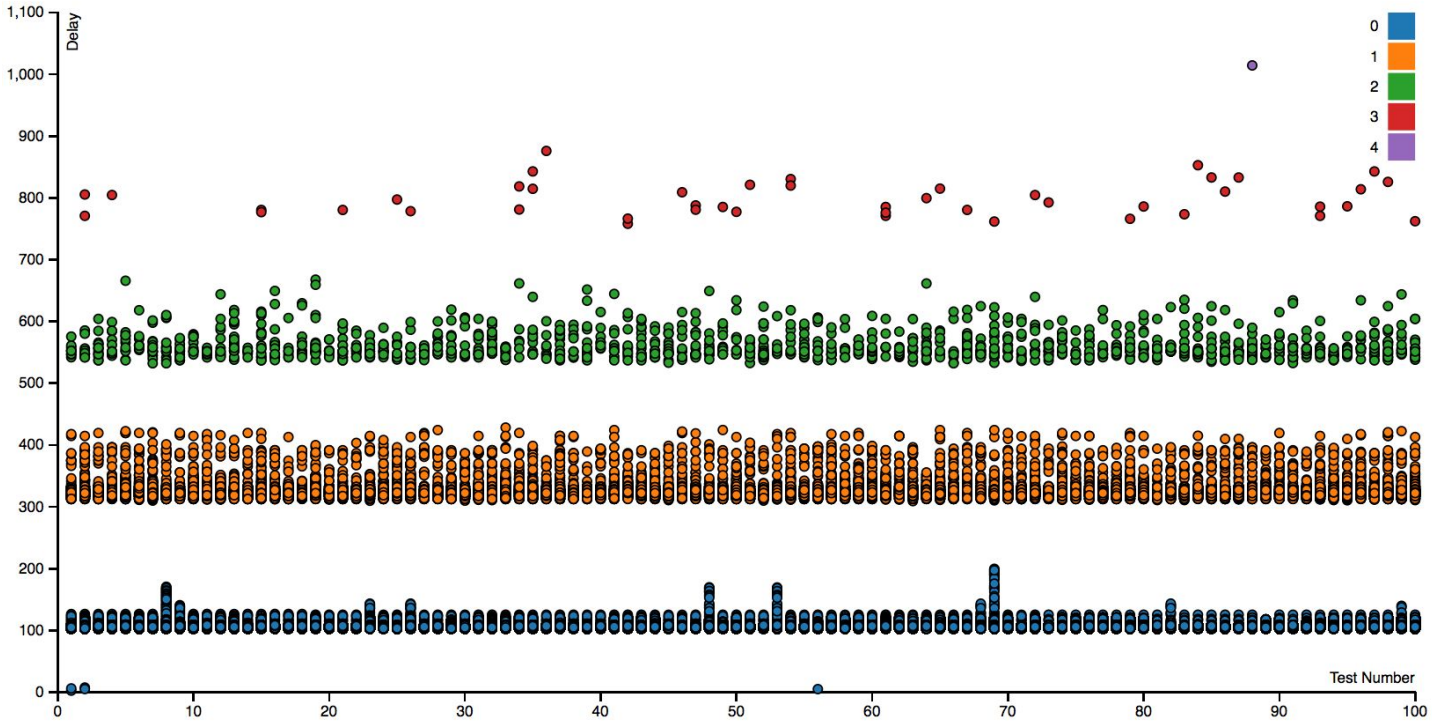

Figure 1: 6Mbps Lateness Scatterplot

DATA PIPELINE

In our experiment we record the arrival rate of blocks of data including the following five fields:

Timestamp — Time since last block — Full block size — Payload size — Block Number — Original or Retransmission

All times are computed in microseconds. We ran 100 transfers of 30 seconds duration, aggregate the data and calculate the "lateness" of each block (original or retransmission), where the lateness is given as:

Lateness = FinalRexTime — SenderT ime — OneWayDelay,

Where *FinalRexTime* is the timestamp of the Received Retransmission, and SenderTime is the timestamp when the corresponding original block was sent. Thus, the *Lateness* is the amount of time passed between the sender originally trying to send, and the final time the block is successfully delivered to the receiver, less the one way delay.

The *RexCount* is computed by counting the number of retransmission requests for that block sent on the receiver side.

We expect that

$$RexCount \times RTT \leq Lateness. \quad (9)$$

From this dataset, we generate our scatterplots (figures 1, 2, and 3). These scatterplots display test number vs. lateness on a per block basis. Some immediate, but important observations:

- There is no statistically significant correlation between test number and lateness.
- The vertical bands in the data, cluster exactly into the expected regions predicted by the FASP protocol.
- Our predicted probabilities are manifest in the dataset.
- We have no inexplicable delays.

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

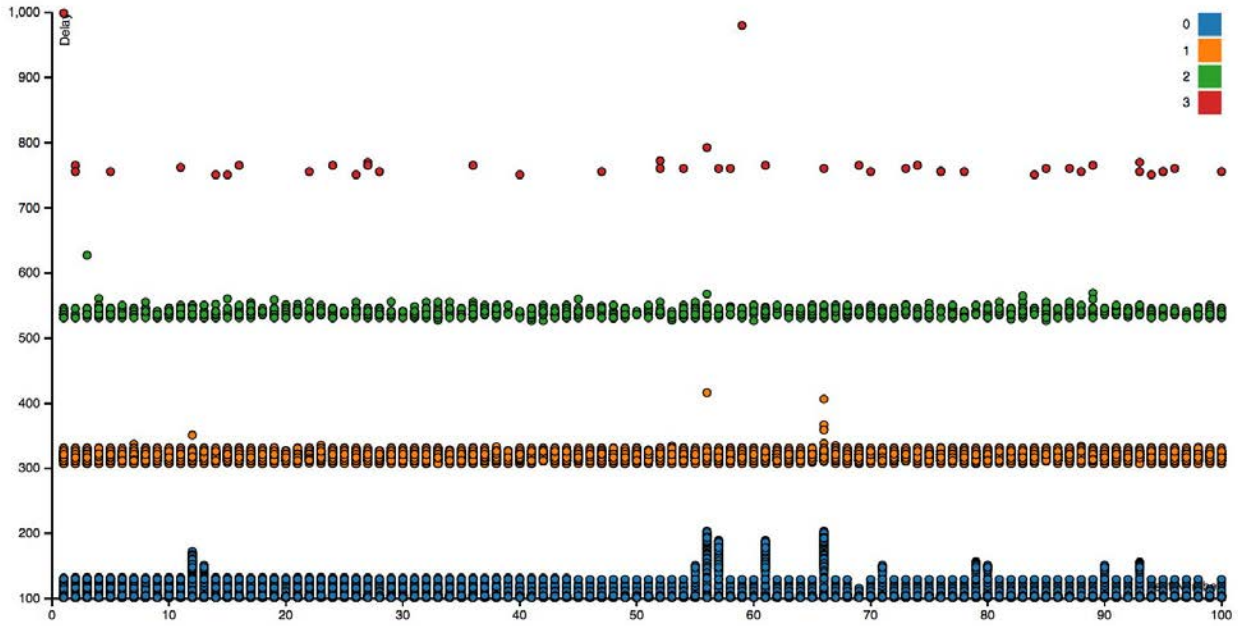

Figure 2: 10Mbps Lateness Scatterplot

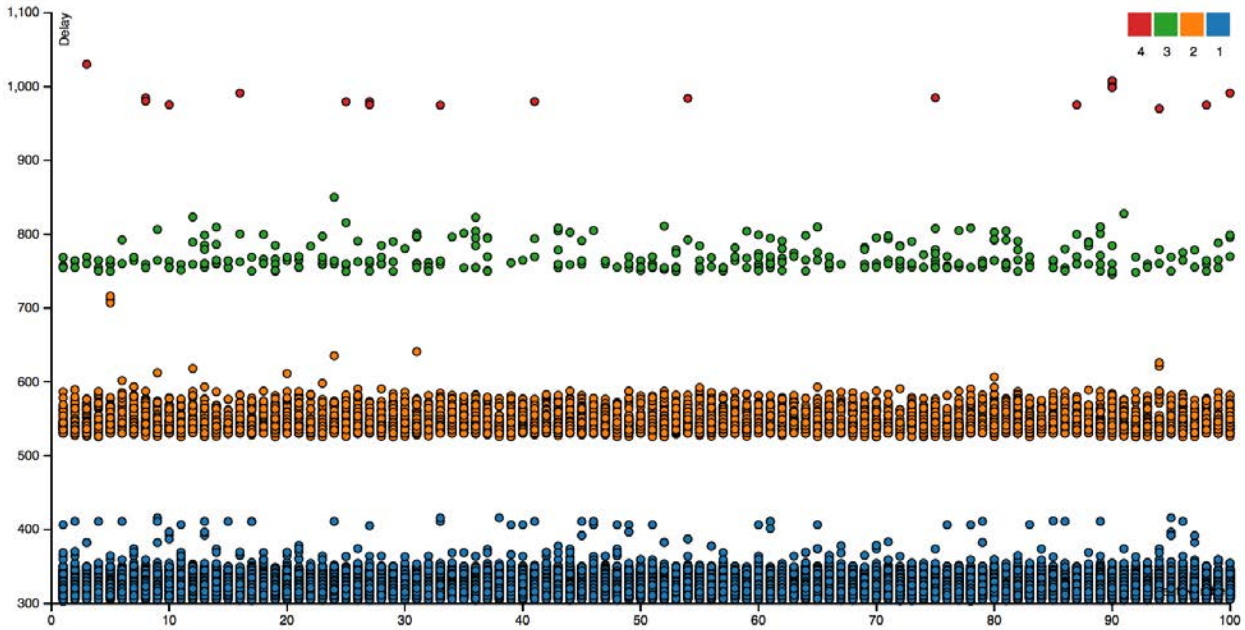

Figure 3: 40Mbps Lateness Scatterplot

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

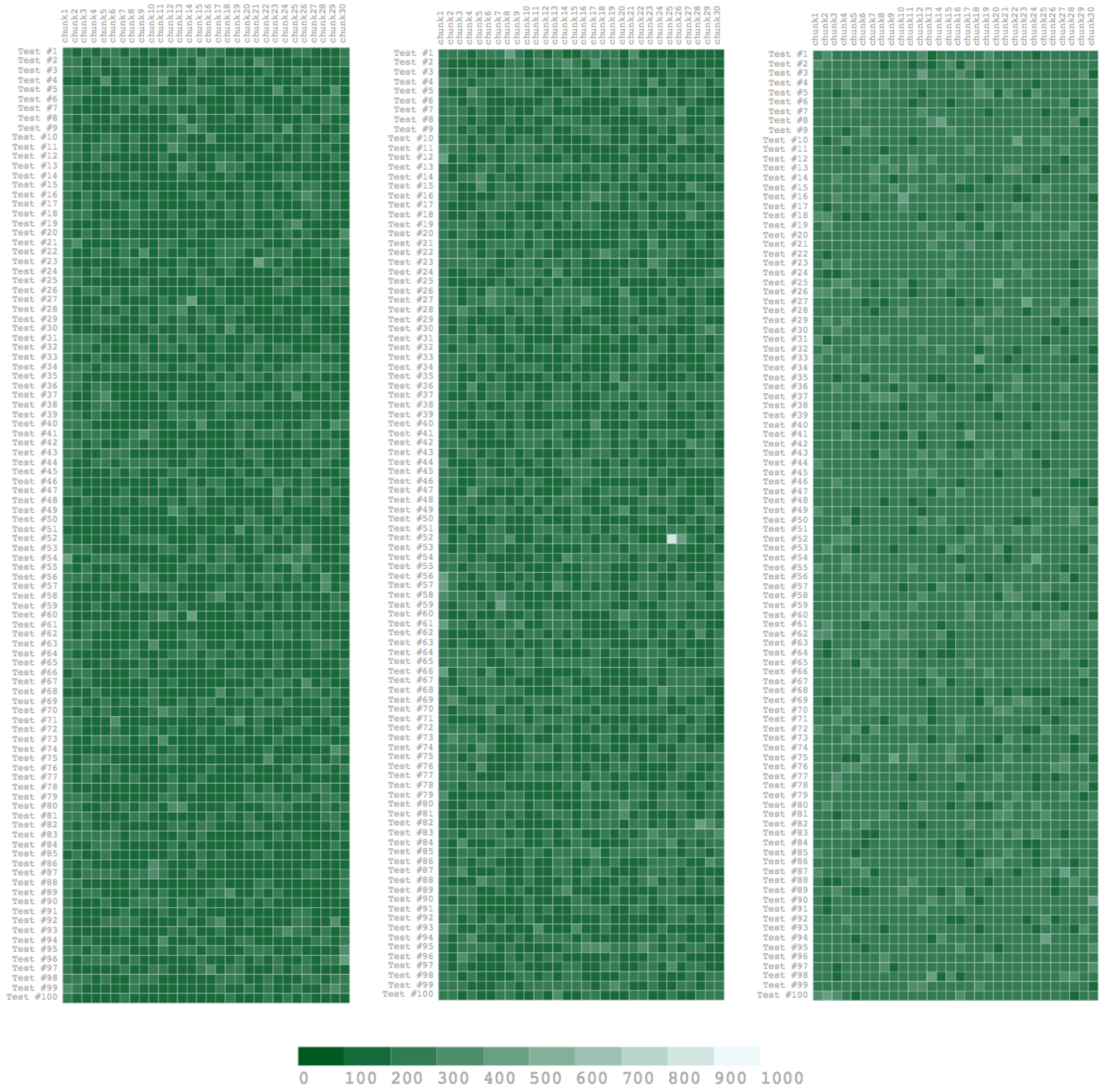

Figure 4: 6, 10, and 40 Mbps Skip Heatmaps(resp.)

Aspera FASPstream for near-live and live video streaming

aspera
an IBM® company

SKIP HEATMAP

Rather than visualize the absolute values of lateness on a per block basis, we want to frame this data in terms observable by users. Specifically, we want to look at the lateness on a per-readchunk level. Humans don't see when a block is late – instead they see when a block is so late that the video skips. The readchunk abstraction bridges us from what we see, to what is happening at the protocol level. Heatmaps are a visual technique for associating regions to values, and are an excellent tool for identifying values above a certain threshold. In our case, we will use white—or "hot"—squares to refer to readchunks that are troublesome. Green squares—or, "cool" squares—designate readchunks that arrive close to the minimum possible dictated by network conditions. Our output will appear as a matrix of squares, where rows correspond to individual tests, and columns indicate the index of the readchunk in the video. We will hope to see primarily green; as the squares grade to white, the corresponding readchunks are arriving later. We now need the data in a very different form:

TestNum — ChNum — Lateness

Again, TestNum and ChNum are as before, however *Lateness* is computed differently. We want to see how long the entire readchunk will take to arrive. We can't simply look at the timestamp of the last block in a readchunk and subtract the timestamp of the first block, because this doesn't respect the possibility that some block retransmissions will come later. So we look at the latest arrival of any block in the readchunk, and subtract from this time the first arrival time. This tells us how long the block took to arrive. We subtract the readchunk's index times the buffer time to convert this value to a comparison to when it was expected.

In symbols,

$$ChunksLastRexTime - ChunksFirstTime - BufferTime \times (ChNum) = ChunkLate. \quad (10)$$

We see excellent performance. In the three test case scenarios we see performance as predicted by the theoretical model. We see in Figure 4 almost exclusively green, and even better, a dark green, close to ideal. Nothing in our testing suggests our model was overoptimistic, and to the contrary, provides us with evidence that we can effectively operate under these assumptions.

BENCHMARKING OUR IMPROVEMENT OVER TCP

The TCP protocol underlying HTTP adaptive bit rate streaming protocols behaves quite differently from FASPstream and is not directly comparable under this model. TCP's congestion avoidance algorithm slows down so aggressively with packet loss that much longer start-up delays are necessary for live streams. For near-live use cases, a huge delay is problematic. To quantify the difference, we show show the kind of startup delay anticipated, for direct comparison to Faspstream.

For the scatterplots(Figures 5, 6, and 7) associated to TCP tests, we plot test number vs. startup delay in milliseconds. The startup delay means the amount of time before the first 1 second bucket, is ready to display. Notice that we can't guarantee continuous playback even from here; this is the minimal time, before we can start playing.

Some immediate take-aways from these scatterplots are:

- The variance in these statistics is extremeley high.
- The values themselves are *huge*, both in comparison to FASP, and for normal use-cases.

As suggested before, delays on the order of 15-120 seconds are totally unacceptable. In any of the cases, we see values well outsides the bounds of what we're willing to accept. Furthermore, we don't even see some cases with good performance; all cases are bad.

CONCLUSION

FASPstream is no more a faster replacement to TCP streaming, than air travel is a faster replacement to American-European driving; *you just cannot drive across the Ocean*. FASPstream offers minimal playback delays, consistent delivery rates, and high network performance with excellent quality including negligible probability of skipping, opening new game changing possibilities for live and near live streaming at distance.

Aspera FASPstream for near-live and live video streaming

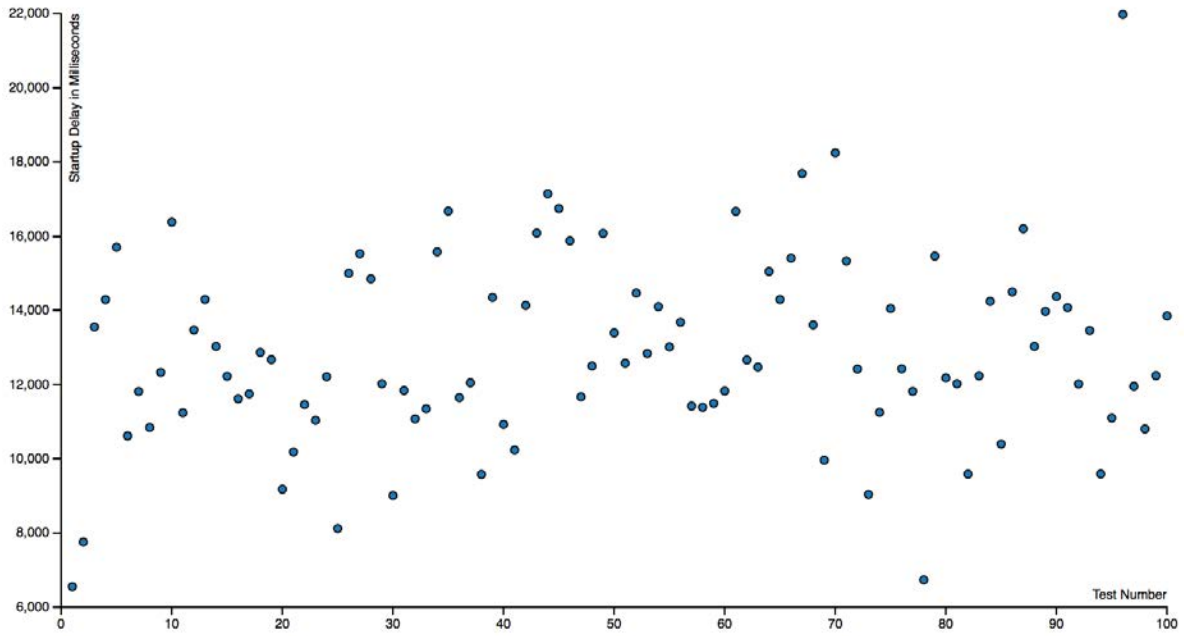

Figure 5: 6Mbps TCP Startup Delay Scatterplot

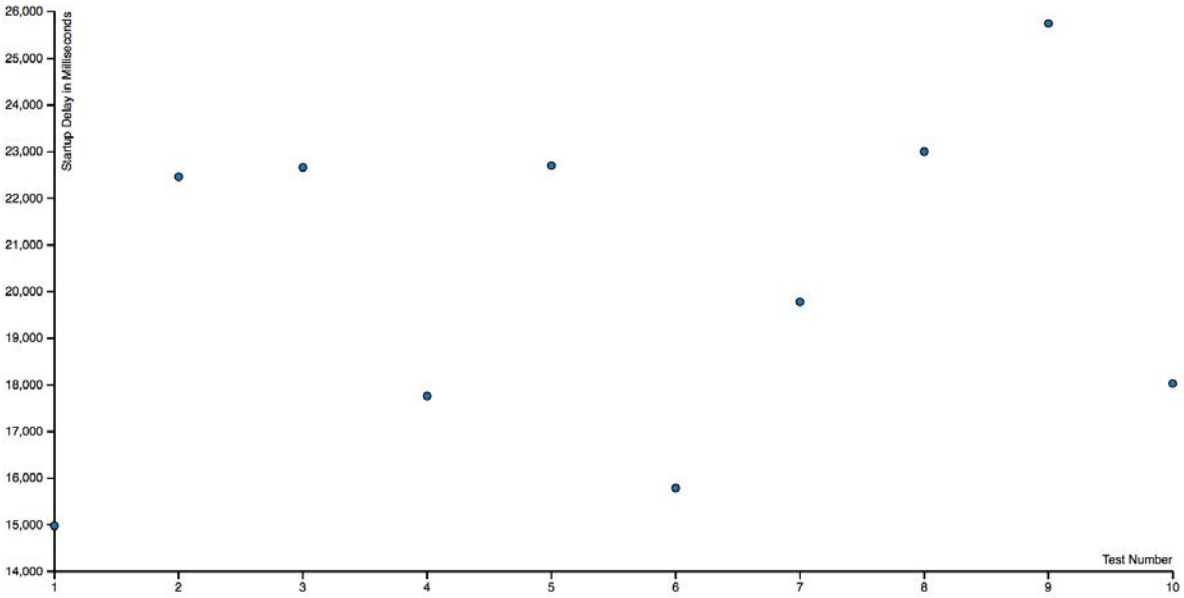

Figure 6: 10Mbps TCP Startup Delay Scatterplot

Aspera FASPstream for near-live and live video streaming

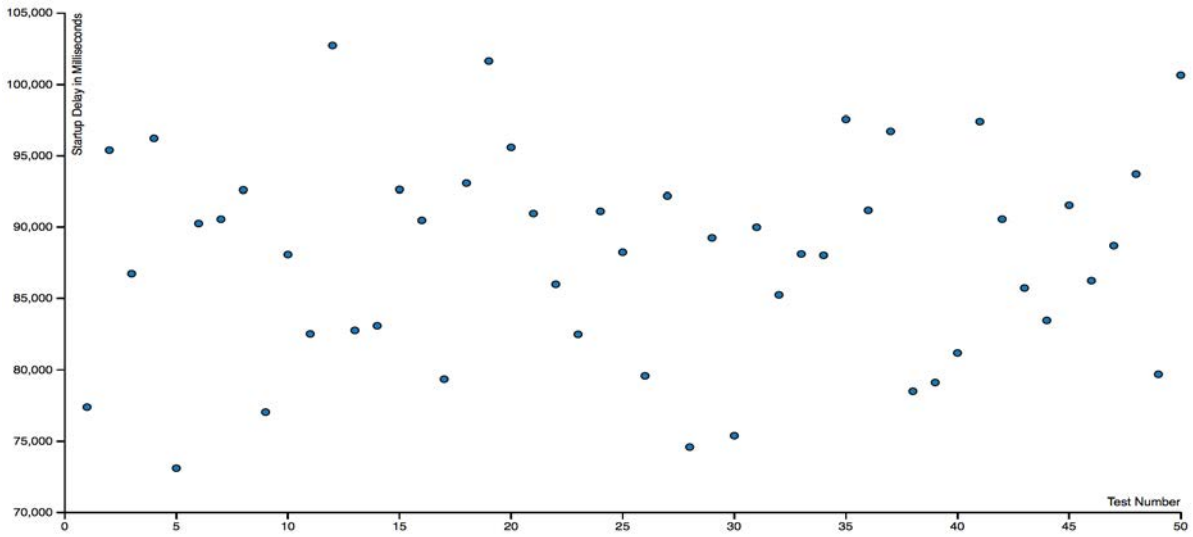

Figure 7: 40Mbps TCP Startup Delay Scatterplot

About Aspera

Aspera, an IBM Company, is the creator of next-generation transport technologies that move the world's data at maximum speed regardless of file size, transfer distance and network conditions. Based on its patented, Emmy® award-winning FASP® protocol, Aspera software fully utilizes existing infrastructures to deliver the fastest, most predictable file-transfer experience. Aspera's core technology delivers unprecedented control over bandwidth, complete security and uncompromising reliability. Organizations across a variety of industries on six continents rely on Aspera software for the business-critical transport of their digital assets.

Learn more at www.asperasoft.com and follow us on Twitter @asperasoft for more information.