Since the introduction of Periscope and Merkatt the landscape is dotted with companies offering so called “livestreams”, i.e. video streams of Real world events such as weddings, conferences, Town Hall meetings and spots events. Most of these streams are available for free viewing but a few companies have experimented with Paid events and value added features available via in application purchase such as that utilizing Apple Itunes store.

When Periscope was conceived it provided a value added feature that allowed subscribers to reply with Live comments to the broadcaster. For business and some technical reasons, we will discuss later only the first 100 viewers of a stream were able to comment all other subscribers could view the stream but without comment. If they wanted access to the higher quality and lower latency streams they had to wait for a high-value slot to become open or pay a value added price to bump themselves in the queue so they would always get a high quality slot.

Latency is the measure of how close to real-time a streamed frame appears in your player, very low-latency means that if you place your figure in front of the camera your viewer will see it in a second or less. Obviously doing real-time commenting requires very low latency streams, whereas for those only viewing the stream a delay of 20-30 seconds may be acceptable.

Some apps allow a quick inapp purchase to upgrade from the free or cheaply priced streams to value-added streams.

The real-world metaphor for this type of stream monetization is a theater or a Las Vegas Show.

When a customer goes on the website for Cirque de Soleil, Chris Angel or any popular Vegas show they are met with an ale-cart menu of possible seats, seats closer to the stage or with better viewing angles are higher priced than the more plentiful general public row seats.

This is a common model for monetization that has not been fully utilized by Live Event providers, i.e. multiple revenue streams (no pun intended) from a single Live event.

This is an attempt to define a new monetization model the “Theater model” We refer to it as “Theater” as it mirrors real life productions. We will present a few existing apps and move our discussion to a hypothetical app we refer to as LiveCaster, which allows subscribers to view real life events such as a baseball game or Olympic sholem at several different price points , with each price point offering a value-added viewing experience.

Facebook Live went public on January 28, 2016 Facebook Live allows users to set up Groups, and designate an individual within the group to be a broadcaster, subscribers to the group can than view live streams. A lesser known Canadian company Conx2share allows users to set up groups and like Facebook-Live allow for the designation of a

User, the owner of the group to Livestream to everyone in the group.

Conx2share frequently streams Canadian Snowcross and has agreements with RockStar Energy drinks to Livestream Bmx nationals.

Conx2share allows a User to create an Event, the owner of the event becomes the broadcaster and can Livestream to anyone in the group who subscribes to the event. Both Facebook live and Conx2share in this way provide some measure of security and privacy.
Streams vary in the quality of the videos produced by the smart phones, Conx2share provides the ability to stream from high definition and expensive video cameras offered by vendors such as Sony and JVC. A few years ago Wowza the most popular of streaming server's providers introduced its "works with Wowza" specification which provided an SDK camera vendors could use to stream directly to the server, no smart phone needed. Auto manufactures and security camera services have been utilizing high def cameras over smart phones for a few years but it is still relatively new to social media.

So it would seem that services such as Facebook Live, Conx2share and Periscope and all its clones would be the de-facto model for streaming but how does it stack up against a real "brick and mortar group"

Birdie is a hypothetical Audubon group; it meets once a week at the local community center at which time members show videos of their encounters with feathery friends. One day a new member John joins the group, John is a professional photographer and introduces the group to all his "toys", high priced cameras.
The Group has access to all these new capabilities as long as John stays with the group, when John leaves he takes his toys with him.

So How do we logically model this with something like Mongo or mySql, as mentioned Facebook Live and Conx2share allows users to create groups, members can be added to the group, members can create events, the owner of the event may broadcast using her smartphone or digital video camera.

From a software architecture perspective, it becomes clear that to mirror Birdie we must abstract out the camera which may or may not be the smart phone.

A first cut might be to abstract "camera" out as a user, so now every digital phone or smart phone becomes a user that can be added to a group or event.
But there is a problem with this. Most service providers desire to collect Analytic metrics on their users such as demographic information. Having all these fake users make this cumbersome to say the least.

As in our Bird Watching example cameras are not much use without users, a real person has to either operate or set up the camera.

So it would make sense to abstract camera into another structure associated with User. A disassociated Entity named camera which may include the user's smartphone can be added to Users.
This model is a pretty close visualization of Birdie, a user may create groups, within groups she may create events and invite subscribers to the event, one subscriber may be designated the broadcaster and a Camera Entity assigned to this User. The "Camera Entity" may have one or more real world devices assigned to it.

This not only provides a nice virtual representation of Birdie but also more closely matches the real-world, for instance a Facebook Live subscriber joining the group from a web browser may not have access to any Smart phone or digital camera. So they realistically can't be a broadcaster A model like this makes life easier for the UI designer, now the application simply needs to check a "can stream" flag on the User entity to determine if the application presents the Broadcast button or not.
This model works well for Analytics as well, metrics can be taken on the group, the event, Viewers and Broadcasters without worrying about the complication of "fake" Users.

But the astute User will notice there is still a problem with this model.

LiveCaster infused with Angel funding sends the astounding John to PyeongChang to cover the Olympics, to save John the excitement of customs we send a backpack of cameras including the all
Important FLIR camera for night shots to meet John…. Yes, Mr Customs agenst sir, It's a Backpack of cameras. yes, backpack…. I know what it looks like…바보

Now John is a master contortionist when it comes to juggling multiple cameras but even John can't be in two places perhaps miles apart at the same time.

LiveCaster adds one further Abstraction to our model, another disassociated entity called camera crew.

A cameraCrew like a real camera crew for a newsroom can be assigned users, cameras can be assigned to Users. Logically it doesn't make sense for a user to have more than one camera but in practicality all users likely have one camera a smart phone along with any additional digital video cameras. So LiveCaster adds a field, Primary camera which designate the primary camera this user will use for broadcasting.

CameraCrew entities are treated like Users, they can be added to groups and events. For purposes of Analytics metrics can be gathered on CameraCrew and also on each individual member of the camera crew.

Meanwhile in Korea. John can assign his camera crew to different locations around Olympic event, including a hilltop overlooking the Grand slaloms.

LiveCaster logically multiplexes all streams coming from camera crews which provide for some interesting app features. Streams from different camera angles or locations can be paged through on a player or displayed in grid fashion. A single view in the Grid can be expanded to full screen. The viewer can experience the event from as many different camera angles or locations as there are physical cameras set up, including that FLIR .

All Streams are not created Equal

Back in the beginning of this paper in our discussion of Periscope and Merkatt we mentioned there were physical reasons why commenting and low latency streams were restricted to the first hundred or so users of a stream.

The vast amount of Users of Periscope, Conxshare and LiveCaster view streams as HLS, Apple developed Http streaming to be an advanced highly scalable protocol for streaming but even Apple will admit it is not very useful for Low Latency Live Streaming. HLS is a segmented protocol, i.e. There is a header file called an M3U8 playlist and multiple segments, there is considerable buffering at the start of a stream and at the start of each new segment, latency to real-time can be anywhere from 20 – 30 seconds, This is fine for recorded events but bad for security or Baby monitors. Just think about what your toddler can do in 30 seconds of unobserved time. On the other hand, HLS is a modern protocol that can easily scale to half a million or more concurrent users.

Latencies as low as 80 ms can be achieved over fast fiber or 5g networks using custom protocols that take advantage of hardware encoding/decoding and using the popular GStreamer with a proprietary format of h264 NULU's streamed over UDP its possible to get latencies of 320 ms even over Wi-Fi.

But most low latency streaming over the internet uses either RTMP which is based on adobe flash and has a Latency of around 7 seconds and RTSP which has a very respectable latency of 1 second or less to real-time.
Real-time streaming protocol consists of audio and/or video streams and a control channel stream.

HLS has the advantage that all Apple mobile devices and many Android phones can easily play without need for specialized players whereas RTSP and RTMP require special players using either mobile SDK's or the open source FFMPEG.

The idea is you put all your value-added commenting customers on the lower latency streams and the rest are re-directed to the higher latency HLS streams.

You pre-sell these most valuable seats for the events so you know exactly what resources you need the day of the event, everyone else going to the loadbalanced lower quality streams.

With this knowledge in hand we now look at the Architecture an app like LiveCaster would use.

The Architecture of LiveCaster

The secret used by the vast majority of streaming services to be able to ramp up for major events and ramp down for more normal optimization is cloud Hosting.

https://www.wowza.com/products/capabilities/stream-scaling

Google, Amazon and Microsoft each provide cloud hosted VM's which can be easily built up or torn down to support a particular event. A major event such as the Olympics could easily require 320 virtual servers over the course of the event. Microsoft Azure and Amazon EC2/AWS are the most popular to use with Streaming Servers such as Wowza. Amazon AMI's can easily be brought up using Cloudformation or Terraform scripting and torn down equally as fast after the event.

A typical Load Balancing scheme for a Wowza Streaming Engine consists of one or more origin servers that ingest incoming streams and x number of Edge server / repeaters that both distribute the streaming load and provide replication of the stream across regionally distributed servers.

A given stream can be accessed from the Origin, or from any of the edges. The Wowza load balancer re-directs the user request to the least used edge in terms of number of concurrent streams, server load and can be configured for best geographically available If LiveCaster has an Origin server in US East coast and two edge servers one located West Coast Another Mid-West, any given stream should be replicated across all three servers, Add one for Asian Pacific and you have most of the world covered.

If the reader thinks the introduction of these edges into the architecture adds latency to the system because the stream to the origin must be replicated to the edges you would be correct.

if we assume that the origin server that is ingesting the input stream operates at O(n) than duplication of the stream across the edges is at most O(n^2), since every byte is duplicated.

This means that the highest value seats, or the best seats in the house are on the Origin servers, if we pre-sell our value-added seats we know just how many origin servers we need for the event.

We can still serve low-latency streams in the >1 second range on the edges, this would be equivalent to 2nd and 3rd row seats in a Las Vegas show.

So What does this mean ?

1) Any stream is available directly from the Origin using the live application or any of its variations, these are the highest quality connections because they have a runtime of O(n) , the input source and output stream being on the same server.

2) Any stream may be accessed from any available edge using the liveedge application or any of its variants, the edges are for volume , they have a runtime of O(n^2) as each source packet is copied to buffer on the edge. massive quality of lower grade streams.

3) Querying the load balancer will select the least used edge in terms of connection count and CPU bandwidth, the load balancer will return this edge to compatible players

Wowza has published some stats for a reasonably configured Amazon AMI and its streaming engine

assume a reasonable latency for HLS when connections exceed 100 to be around 35 - 40 seconds.

Assume that Wowza can only serve only 5 Gbps,

Some customers have been able to get up to 10Gbps of streaming performance for both live and on-demand on multi-threaded, multi-CPU machines that were finely tuned at the kernel level. On standard servers with dual quad-core processors and multiple NICs, up to 5Gbps of streaming performance can be achieved if the server is properly tuned. Tuning guidance is available in our Performance Tuning Guide.

Going off the stream bitrate (600 Kbps) and the concurrent connections at peak time (50,000 users), you will need around 8 servers at this specification.

600 Kbps x 50,000 users = 30 Gbps which is impossible with one server limited a maximum of 5 Gbps.

Realistically you're looking at 6,650 connections per server assuming a 5 Gbps connection is available, giving a 20% overhead.

600 Kbps x 6,650 = 3.99 Gbps

50,000 /6,650 = 7.51

50,000 = Total users, 6,650 = Connections per server and 7.51 = Servers needed.

Stream Monetization

With all we have learned we can now come up with a reasonable monetization model for LiveCaster Streams. LiveCaster would make the value-added content available via in-app purchase on a per-event basis.

Free - 0.99 - best value , 30 seconds time shifted, unobstructed view high def , whatever server the load balancer selects.

2.99 best user - experience , instant as it happens, high def, send comments while streaming, 30 second instant replay

4.99 Prime location - hill above the event, best viewing angles, like having a camera behind home plate at a baseball game.

6.99 - Prime experience - all of the above but includes value added comments such as keepsake download of the stream.

And there you have the Theater model, we monetized a single stream into 4 price points, and with creativity we may have been able to add a few more How about streams with live celebrity commentary for instance.