**CX-1094C**

# Overview of MoveNetwork's Streaming Media Technology

V1.0

## Introduction

The new streaming media technology invented by MoveNetworks is comprised of several innovative techniques. In totality, these techniques provide a dramatically useful and new approach to the real time delivery of media content over computer networks, especially the Internet. There are many pitfalls in current technologies that result in poor user experiences. The technology we have invented is named "Quantum Streaming"; it is designed to solve or mitigate most of the pitfalls found in the current streaming art.

This document attempts to present in basic terms the innovations in Quantum streaming by defining the technological context and challenges to be overcome and then explaining how Quantum streaming provides novel solutions to those challenges. In some cases, the inventions actually extend to capabilities previously unconsidered for the streaming media realm.

The author claims sole responsibility for any errors or ignorance expressed herein. Corrections and additions are welcome.

## Streaming v. Download

"Streaming media" refers to technology that delivers content at a rate sufficient for *presenting it to a user in real time as the data is received* and (traditionally) not storing the content on the user's hard disk. The user has immediate satisfaction at viewing the desired content. However, the audio/video quality that can be received for real time presentation is constrained by the available bandwidth of the user's network connection. Streaming can be used to deliver content on demand (previously recorded) or from live broadcast.

In contrast to streaming, media content can be downloaded as a file. Downloading complete media files takes time. Once downloaded, however, the content can be viewed repeatedly anytime, anywhere. Media files prepared for downloading usually are encoded with higher quality audio/video than could be delivered in real time. For example, a high quality movie trailer for *LOTR: Return of the King* that I downloaded had a file size of 20MB. Even with a 640Kbps DSL connection the download takes about 6 minutes.

(A variant technology called "progressive download" initiates a file download and then waits until enough of the file has been downloaded that playback can begin with the hope that the remainder of the file will be completely downloaded before playback catches up. The delay before playback can begin is usually substantial. For example, if a two minute

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068029

media file takes eight minutes to download, playback will not begin until six minutes have elapsed from the start of the download.)

Streaming offers the advantage of "immediate" access to the content but with a sacrifice in quality compared with a downloaded file of the same content. Streaming provides opportunity for a user to select different content for viewing on an ad hoc basis, while downloading by definition is restricted to a specific content selection in its entirety, or not at all. Streaming supports rewind, fast forward, direct seek operations poorly or not at all, especially with live broadcast material, but a downloaded file can be navigated at will. Streaming is vulnerable to network failures or congestion along the delivery path, whereas a media file, once downloaded, has no dependencies on network conditions (or even having any network connection at all.)

Any user who has viewed a streamed video clip over the Internet almost surely has experienced the shortcomings of the current art: several to many seconds of delay before playback begins, hiccups, video going into "slide-show" mode, total freeze with "buffering…" messages dutifully reported by the player. The technical causes of these failures are several. Generally, they can be categorized as inadequacies in coordination between media servers and client players, transport reliability, adaptability to adverse network conditions, and scalability for handling high numbers of users.

Enter Quantum Streaming

At MoveNetworks, we have devised a streaming architecture to dramatically alleviate most limitations of media streaming as compared with the current art. To that end, we have designed new approaches to content preparation, origination, transport, and control logic. We have dubbed this framework "Quantum Streaming."

One core aspect of our technology is the concept of using *streamlets*: streamlets are the result of dividing a video stream into sequential pieces of some arbitrary but consistent chronological length. In concrete form, a streamlet may vary in size or format from its fellow streamlets, but the playback time values are the same. The chronological length of streamlets is 2 seconds in our current implementation. (The reasons for this choice are explained later.)

Splitting files, data objects, merchandise, or information into pieces of various sizes for transport over "networks" is a fundamental and well-known concept in all areas of life. A transport caravan of five camels, each with a 500lb load, works better than one camel with a 2500lb load. [The latter only gets you an unreliable and terminally non-scalable delivery network for your spices. Flattened camels neither adjust their rates nor perform fast startup operations.] If one camel is stolen, you at least have four camels to work with while you order a replacement.

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068030

The use of streamlets combined with our other innovative concepts makes possible:

(1) Fast startup of playback when selecting content with automatic rate selection,
(2) Adjustment of the streaming rate for a client up or down very quickly in response to changing network performance conditions,
(3) Enhanced Rewind, Fast-Forward, Scrubbing (dragging the sliding position indicator to select the playback time index),
(4) Instant jump to related content,
(5) Reliable and efficient transport of content,
(6) Elimination of specialized media servers and their attendant management,
(7) Containment of all content selection, seeking operations, and rate adaptation logic to clients (as opposed to a combination of client and special media server),
(8) Radically enhanced, network-wide scalability.

The streaming of media content over a network, such as the Internet or an office LAN, involves four supporting components:

(1) Content preparation
(2) An origin point of the content
(3) A mechanism for transporting the content over a network
(4) A receiving point to process the content, usually with the intent of presenting it to an actual user.

The following sections will describe how the use of streamlets and other innovations are applied to these four components.

Content Preparation

When media content is prepared for streaming, an original very high-quality source is encoded into one or more lower-quality representations suitable for transporting across ordinary networks for real time playback. Examples: 100Kbps, 200Kbps, etc. Practically speaking, higher encode rates represent higher quality audio/video. The rate is a measure of how much bandwidth a user's network connection must have available to sustain real time reception and playback. Encoding multiple rates allows for supporting users with a wide range of available bandwidths. Most streaming architectures are not able to switch among the available rates once streaming has begun.

Streamlets can also be encoded at multiple rates. Because every streamlet has an identical playback time to any other streamlet, and starts and stops on the same exact time boundaries as its cousins, it is possible to make on-the-fly substitutions between streamlets that cover the same time space in the content but which have different encode rates. (There actually is no reason why streamlet durations could not vary from one to the next in their chronological sequence, as long as the cousin streamlets (at other encode

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068031

rates) also aligned with the same start and end points. Consistent sizes makes implementation a bit easier.)

Content Origin Point

In practice, streamlets are encoded into files, one file per streamlet. The streamlets are then stored where they are accessible by any industry-standard web server that requires no modifications. Streamlets are retrieved through standard HTTP GET requests; streamlet path and file names are selected to uniquely identify each streamlet's time sequence in the content as well as stream rate.

All other streaming solutions available today rely on:

1. Specialized media servers.
2. Specialized protocols, (even if using HTTP as a carrier protocol).
3. Coordinated exchange of control logic information between a client and the specialized media server. This logic covers at least the selection of stream rate (if supported) and the position within the content being selected for playback. (As an example of rate selection, the Windows Media 9 Series (MMS) server decides what stream rates to send to each connected client.)
4. A direct connection with the origin server. Therefore, the media server must be supported by a very large infrastructure to support many clients and is not very scalable.

Quantum streaming eliminates all of those elements and in comparison provides significant improvements in capability and performance over the other solutions.

A. An ordinary web server becomes the origination point for the quantized content. No software installation or management software is needed to serve the content. (Consider the hundreds of pages of documentation to operate a Microsoft Media Server.)
B. The only protocol employed is an industry standard HTTP "GET" method for requesting a web object from a web server. No custom or dynamic responses are required.
C. All control logic for rate adjustments and content seeking occur in the client, eliminating the need for a specialized server.
D. The streamlets retrieved from the web server can be cached by industry-standard web caching devices. (Dynamically generated responses usually are not cacheable. And a dynamically generated response implies special handling and customization to process the custom requests.)

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068032

Content Transport

There are three basic challenges with data transport in a request-response environment:

(1) Reliability.
(2) Efficiency.
(3) Latency.

We'll now define these three issues in general terms and then explain how quantum streaming solves them in comparison to other streaming solutions.

**Definition of Reliability**

Most other streaming solutions use a TCP connection, or "virtual circuit," for transmitting data. Some allow for use of UDP. A TCP connection provides a "guaranteed delivery" mechanism such that data sent from one endpoint will be delivered to the other end, even if portions are lost and retransmitted. UDP provides no connection or delivery guarantees whatsoever, shifting any reliability handling upwards to the application. A break in the continuity of a TCP connection can have serious consequences when the data must be delivered in real-time. Network failures and congestion are a native hazard on networks, especially the Internet. When TCP detects delays or losses, it immediately "backs off" from its transmission attempts for a moment and then slowly resumes its transmission pace. This behavior is an attempt to alleviate the perceived congestion. Sometimes a TCP connection will simply get "stuck" for some large amount of time (many seconds) or actually give up. Software at each end of the connection may not know of these failures for quite some time.

**Definition of Efficiency**

Efficiency refers to how well the user's available network bandwidth is used for delivery of the content stream. This measure is directly related to the reliability of the TCP connection. When the TCP connection is suffering reliability problems and is not transmitting data for whatever combination of reasons, a loss of efficiency in bandwidth utilization results. A measure of efficiency is relative to the available bandwidth; the bandwidth available to a user's network connection can and usually does vary, sometimes, suddenly. Available bandwidth is affected by other users on the same network (if sharing an Internet connection), other activities by the user's machine, and the performance of the slowest network link between the user and the origin server.

**Definition of Latency**

In request-response architectures, latency is the time measure, from the client's point-of-view, of the interval between when a request is issued and the response data begins to arrive. This value is affected by the network connection's reliability and efficiency, and

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068033

the processing time required by the origin server to prepare the response. A busy or overloaded server will take more time to process a request, if it even can.

Quantum Streaming Transport Solutions

With a single innovation, Quantum streaming solves or alleviates to the extent possible, the problems of reliability, efficiency, and latency. The secret lies in our use of multiple TCP connections between the client and the origin web server (or web cache—the intervention of a cache is transparent to the client.) Other streaming solutions use a single connection and receive their data serially. By requesting more than one streamlet at a time ("parallel retrieval") with each request on a different connection, efficiency is raised to near maximum and latency is virtually eliminated.

Any single TCP connection, as explained, can suffer delays and interruptions in data flow resulting in wasted bandwidth. Under optimal conditions, if two connections are in use, the flow of data on each connection will share the available bandwidth roughly 50/50. In practice, there is seldom such perfect balance. At any given moment, the bandwidth not in use by one connection is automatically consumed by the other connection. Maximum efficiency in delivery of the content is achieved. In our current implementation, we allow a maximum of three outstanding requests (each using its own connection). To further mitigate efficiency loss in the event that a given connection fails, we maintain two "spare" connections. Requests are rotated among all of the open connections to keep each connection's TCP flow logic from falling into a slow-start mode or deciding to close.

When a TCP connection fails completely, or the client has waited too long for a response, a new request can be sent on a different connection for the same streamlet. If the first response suddenly arrives, the redundant request's connection can be aborted. Note that because every request is made on an independent connection, the completing or aborting of that request has no impact on the success or state of other outstanding requests.

There is a general technique called pipelining often used when multiple requests are being made of a web server (such as for loading a web page). Several requests can be sent on a single TCP connection and the response will flow back in matching order along the same connection. This technique eliminates all but the first request latency. Imagine sending all your camels to the bazaar, one by one, with mutually unique orders for certain goods. The camels are loaded up in a first come, first served order. As the first camels are returning, the subsequent camels are being loaded. Eventually your camels return to you, one after the other. Only a single latency was experienced. There is a serious vulnerability to pipelining: if the first camel is shot by robbers on the way back, a replacement for its unique cargo will take a while unless you shoot all the other camels to make way for the replacement. When streaming media for real time presentation, not any old camel will do: you need that first camel! Even worse, because TCP guarantees in order delivery, ALL the camels behind the one that was shot are stuck in place. Only by reviving the one that was shot can the flow of camels resume. If the first camel is actually dead (i.e., the connection is stalled for a long or indefinite period of time, or broken), All

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068034

the camels must be shot and their cargo reordered. To use another metaphor, a failed TCP connection results in a train wreck. All of the different responses are lost. Putting the content delivery back on track is costly in efficiency and presentation time that the player might not have to spare. A poor user experience can result.

By making individual requests on separate connections, Quantum streaming innovatively reaps the benefits of pipelining without the train wreck vulnerabilities. Because multiple responses are always being transmitted, the latency of each new request is irrelevant (and, for the curious-minded, generally not even knowable.) Camels are always running to the bazaar with their unique streamlet orders while loaded camels are returning. Even though each request (camel) is actually using a different connection (a parallel lane in the road between caravan and bazaar), the latency-eliminating benefit of pipelining is achieved. We call this technique "virtual pipelining." It is made possible only with a combination of streamlets and parallel connections.

The same parallelism holds the key to mitigating the vulnerabilities of single-connection pipelining. Because your camels are not required to travel single-file, the loss of one does not require terminating the other camels. A new lane on the highway is opened for use by a replacement order. Furthermore, the replacement order does not have to wait behind other camels but can return directly in its own lane. The replacement arrives quickly and possible delay in media presentation is substantially mitigated. Because the other camels have already returned, their streamlet cargoes are already available for subsequent use without the need for re-requests.

## Client-side Streaming Control Logic

A key and *far reaching* conceptual and practical innovation of Quantum streaming is the *elimination of the need for specialized media servers*. All other streaming solutions depend on various methods or protocols, whether standardized (like RTSP/RTP) or proprietary, to exchange control information between the client and the server. This requires that a connection (TCP or other protocol built on UDP or IP) be maintained between the client and the media server. Decisions about what streaming rates to use, where in a piece of content to begin playback are good examples. The control connection may or may not also be used to transmit streaming data. Just as a data connection is vulnerable to reliability, efficiency, and latency considerations, a control connection is too. Failures can result with mild or severe implications for player operation and user experience.

Quantum streaming addresses all of these issues and adds enhanced abilities in content selection and navigation without a specialized media server and without any kind of control connection with any entity outside of the client. Please note that there are enormous and innovative implications for scalability that are discussed in the next section. Tying a client to a server for the duration of a media experience is an enormous liability. If we can stake our claim to "server-less streaming"…

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068035

With Quantum streaming, the client is always on the watch to use an appropriate stream encoding rate selection so that the available bandwidth is not exceeded by the downloading of the component streamlets. The basic question is not, "what is my available bandwidth?" but, "*am I receiving the current stream fast enough to present it in real time?*" The encoding rate of the content stream is, in and of itself, irrelevant to answering the question. If the client observes that over some modest interval of time (say 20 seconds or so) the player is, on average, consuming streamlets more quickly than they arrive, then a stream of lower quality must be selected. *Because the client can select and retrieve directly from an array of streamlets of different encode rates*, there is no need to inform any media server and ask it to change the data it is sending. (In some solutions, (Like the Microsoft Media Server) the server monitors throughput and makes the decisions on what stream rate to send to the client.) Further details on the adaptive rate-shifting techniques for Quantum streaming are found in a different document. This innovative flexibility in quickly adapting to changing network performance is made possible by the streamlet architecture and eliminating the need for specialized media servers.

The ability of the client to directly select among any available streamlets has other substantial implications. First, changing the time position for playback is trivial: the client simply requests streamlets corresponding to the new time index into the content and continues requesting streamlets from that point forward. Interference from outstanding requests can be eliminated by aborting those connections to speed the retrieval of the first new streamlet. Second, the first streamlet(s) retrieved when changing position can be of a lower (and therefore faster to retrieve) quality than the streamlets currently being used as the "steady state" or "best sustainable" rate. This results in a smaller latency before the user begins to see the content at the different time point. Streamlet quality for subsequent selections can then be ramped back up to the best sustainable rate.

As a practical implementation note, we cache streamlet files on the user's local hard disk as we receive them. This allows them to be reused if the user requests a change in playback position (such as a Rewind or Scrub bar movement) and the streamlet is needed in the course of playback from the new time position. However, there is more than a practical implication to local caching of streamlets—it makes possible further innovative features.

The client's ability to directly seek into a specific time and quality space of the content is more significant that it might appear. Direct requests for streamlets far in advance of the current playback position can be made and the streamlets saved in the cache. If the user wants to fast forward (by any mechanism), the player will find content preloaded at defined intervals (say 30 seconds) from which it can draw to provide an instantaneous change in playback position for the user. Preload requests can be intermingled with other streamlet requests issued in the normal course of playback.

Extending this ability further, with relevant information from a web page, scripting information, database, or elsewhere, the player might be aware of (1) specific index

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068036

points within the content or (2) other related content. Intra-content index points might be scene changes, sports plays, or news stories. Related content might include other episodes or broadcasts of the same program, content that matches searches by subject, actor, or other metadata, or that follows certain chronological time lines. Along any known association, the player can preload the first streamlet from the given "jump point." The moment the user makes a selection, the new content begins playback instantly. This would simply not be feasible with other streaming architectures that are geared toward serialized streaming of media, not random access. For a player to negotiate many connections to many specialized media servers at once in hopes that any particular connection might be used is not reasonable, and even if the player could, what would it retrieve? Playback at any jump point is trivial for any quantum streaming based player. Note that it is the streamlet architecture that makes this ability possible, not any particular player.

Scalability

Streamlet files are simply that: files. They have a static nature; they do not change. Retrieving them involves nothing more than asking the web server to send one. No custom application programming at the web server is required. Frequently, a web server will also host various applications (such as a database manager). Such an application server is handed requests by the web server and generates, or constructs on-the-fly, an appropriate reply. The reply is handed back to the web server and the web server sends the reply to the client. Web caches do not, with few exceptions, cache dynamic responses because any particular response is not likely to ever be used again. It is important to underscore that because streamlet files are static objects they are not only cacheable, but assumed cacheable according to HTTP standards. If a streamlet file is not requested again, then a web cache will soon eliminate it. But if it is often requested, it may stay cached indefinitely.

Any caches existing on the network path to the client will likely cache streamlets so that any near term *reselection* of the same content would be served from the nearest cache(s) and not the origin web server. (Note that most all ISPs utilize web caches at their POPs (Points of Presence) to reduce usage of upstream bandwidth.) This benefits the origin web server through eliminating redundant requests. The user is benefited by the decreased probability of disruptive network conditions due to the shorter request-response transmission path (possibility the other end of their telephone line), resulting in a more reliable playback experience. These are some benefits for a single user case. "Watch once, watch better the next time."

But what advantages are possible with multiple content users? A *unique* characteristic of this arrangement is: *the more users that are accessing the same content, the more scalable the architecture.* More users of the same content result in more caches getting involved which reduces the load on the origin server, shifting the burden to caches elsewhere in the Internet (such as at ISPs). Whenever there are network paths common to

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068037

multiple users consuming the same or recently used content, there is an n-fold increase in scalability at the point where the user paths converge. Obviously, web caching has been around for a while for gaining these benefits for delivery of ordinary web objects, like the Google logo, or little green buttons on some web site. But using them to enhance *streaming media* has *not* been done because other streaming solutions rely on two-way client/server protocols.

This innovative behavior is especially well suited to live broadcasts of significant events; a single web server that responds to just a few thousand requests per second, when logically "surrounded" by several strategically positioned high-performance web caches, might support a million viewers. We shall see!

The very worst use case is when every request to the origin web server is for different content. In that event, performance and scalability are no worse than the traditional specialized media server architecture.

To repeat, these scalability benefits are made possible *only* by using a streamlet approach in the way we have designed: by nature, streamlets are simple, static files to be simply requested and returned. That's what web servers and caches do best, and we reap any performance benefits there may be.

*Copyright © 2004, MoveNetworks. All rights reserved.*

*This document contains proprietary information and trade secrets of MoveNetworks (XLON Technologies LLC). Reproduction or distribution without express written consent is prohibited.*

CONFIDENTIAL BUSINESS INFORMATION DISH_337-TA-1265_00068038