Sharing Big Buck Bunny in XIA

Benjamin Bramble
UW-Madison

Stephen Brown
UW-Madison

Abstract

The Internet has evolved rapidly over the last twenty years, flowing away from early Telnet designs to a more content-oriented nature. Researchers developed XIA to redesign the underpinnings of the Internet to accommodate not only today's Internet traffic, but allow future innovators to add principals as technology changes. Within this architecture we sought to create a video streaming application using the foundation that XIA provides. We present a novel approach to using DAGs in order to facilitate network-level failover for content chunk delivery. We capitalize on the connectionless nature of XIA's chunking protocol by creating content servers that simply host content without any need for connection-handling. We also use a centralized directory to foster the lookup of CIDs based on video name. Our streaming video application is publicly available and should help guide future XIA application developers.

1 Introduction

With the accelerated demand of video services such as Skype and Netflix, video traffic has grown to dominate Internet traffic [8]. The original ARPANET, including Internet Protocol, was designed for survivability and sharing information in an end-to-end manner [2]. The architects at the time built IP around FTP and Telnet, the then-predominant means of transferring files and remote logging. However, by 2010 the proportion of traffic devoted to Telnet and FTP is miniscule, with patterns being dominated by services such as peer-to-peer, video, and web traffic [1]. As the number of available IPv4 addresses shrank, Internet leaders identified the problems with IP and published the IPv6 RFC (RFC 2460) in 1998. Fifteen years later, IPv6 is approaching 1% of global Internet traffic [6]. The slow rate at which IP evolves leaves much to be desired.

Today, IP is still the "narrow waist" of the networking stack that forces all traffic to match the protocol. However, brave researchers at CMU and UW-Madison have been developing a replacement for IP – one which will expand the narrow waist to allow the Internet to evolve to meet changing demands [4]. Hence, they created the eXpressive Internet Architecture (XIA) which, unlike IP, provides for richer network layer semantics that go beyond simple host-to-host communications – this plays well into the current content-based traffic patterns of the internet today.

Few applications currently exist that are able to leverage XIA's strengths in interesting ways. Therefore, our goal was to create a video streaming app that leverages XIA's strengths (failover, content oriented, and intrinsic security of CIDs) to create a product that surpasses what can be done in IP today.

We describe an application architecture that is able to stream video without interruption even in the presence of content server churn. It accomplishes this by utilizing the connectionless nature of XIA's chunk request mechanism combined with the expressiveness afforded by XIA's DAG addressing mechanism. Our applications support two modes of failover - one at the network layer, and one at the application layer. The features of this application clearly surpass the functionality that could be possible in an IP-based network.

Our final application streams the popular video clip from the Blender Institute called Big Buck Bunny. The XIA prototype included this video clip with their example applications so it seemed appropriate to launch it as our demo video. As such, we grew quite fond of the title character and include references whenever possible.

The remainder of this paper is organized as follows: We discuss XIA and other related work in Section 2. Section 3 describes our application architecture and design features. Section 4 goes into our implementation details, describing how we constructed each entity of the architecture. Section 5 describes various scenarios that afforded us a means to test our application in interest-

ing ways. In Section 6, we discuss the central questions raised by our work. In Section 7, we describe the challenges we faced, particularly from our perspectives as application developers. Finally, in Section 8, we describe some recommendations for XIA and in Section 9, we discuss possible avenues of future work. Finally, we conclude in Section 10

2 Background

As discussed above, researchers developed XIA as an alternative to IP. The architecture supports a variety of principal types such as administrative domains, hosts, services, content, and allows for new principal creation to support its goal of evolvability. This eXpressive Internet Protocol (XIP) operates by identifying principals with eXpressive Identifiers (XID) that allow routers to create forwarding tables with entries for hosts (HID), administrative domains (AD), or content (CID) to name a few. When a client wants to connect to another principal, it creates a Directed Acyclic Graph (DAG) representation of its intended destination; in turn the network is able to service the request in a flexible manner. XIA supports the notion of fallback by expressing additional addresses in the DAG in the event that the intent is unavailable at the current XID. This allows XIA to deliver a request based on the client's intent represented by the DAG. We further discuss the manner in which we create our DAGs to support fallback in Section 3.3.

The XIA development team released Version 1.0 of their prototype in mid-March 2013. With it came several example applications designed to copy basic network tools like traceroute and ping as well as some more advanced applications such as file chunk transfer clients. Among these, XIA's demo also included a sample video server, which could be used to deliver video to a Firefox browser with HTML5. We borrowed profusely from the video server code when creating our CID Directory Server and Content Servers, discussed in Sections 3.1 and 3.2 respectively.

While we were able to adapt the video server to our needs, we saw a need to step away from the confines of Firefox and create a stand-alone C++ client application. In turn, this allowed us to implement our rich failover mechanisms. Much of the Video Client borrows from an example chunk copy client that used chunking to receive files from a server. We describe our Video Client in Section 3.3.

The XIA prototype also includes a sample network that runs on Click within the confines of the provided virtual machine. In order to create a richer setting in which to test our applications, however, we created a simple network topology that expanded upon the basic demo network. A discussion of our particular network can be found in Section 4.5.

3 Application Architecture

We designed our architecture to take advantage of XIA's support for connectionless retrieval of content chunks. In the spirit of this, our design consists of simple, barebones content servers which simply host the content in their content caches. In addition, we built a CID Directory Server that maintains a list of content server locations, as well as a list of CIDs that correspond to a given video. The CID Directory Server facilitates client lookup of content locations. And finally, to round out the herd, we include a video client which simply streams the video from the content servers.

3.1 Content Servers

The XIA API enables a new usage model in the design of clients and servers when dealing with content. It allows clients to request content chunks without ever having to establish a TCP-like connection with the host on the other end. In turn, this allows the content servers to be vastly simpler than their equivalent in an IP-based network, as they no longer have to handle client connections and respond to requests. As such, our content servers are very simple: they simply are programs which upload video chunks to their content caches. Since the content caches enable connectionless access to the content, the programs themselves do not need to handle any connections. In fact, they do not even have to remain running.

3.2 CID Directory Server

In order to provide clients with a means by which to locate the video content they seek, our architecture includes a CID Directory Server (CDS). For any given video, the CDS keeps both a list of content server locations (ADs and HIDs) as well as a list of CIDs for the video. If a client connects and requests information for Big Buck Bunny, the CDS can respond with the list of Big Buck Bunny content locations. In subsequent RTTs, the client can request groups of CIDs by specifying entire ranges.

In our architecture, the CDS is a traditional multithreaded server which accepts connections from clients and handles requests. However, in future versions, it could easily be replaced by an out-of-band means such as a website.

3.3 Video Client

The role of the Video Client is quite straightforward: it streams a video from the network and plays it for the

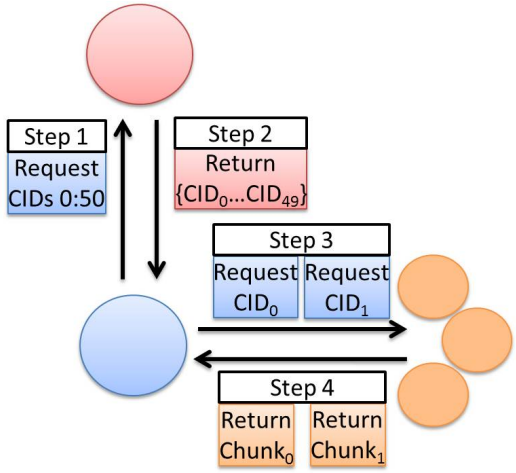

Figure 1: Sample interaction: After setting up a connection with the CID Directory Server (CDS) and indicating what video the client would like to watch, the video transfer will proceed as follows: the Video Client will contact the CID Directory Server to acquire a list of CIDs. The client will then issue chunk requests, which will get serviced by one of the content servers. Finally, the network transports the chunks back to the client, which then adds them to its video stream.

enjoyment of its user. In order to do this, it must first establish a connection with the CDS and indicate that it wishes to stream a video (e.g. Big Buck Bunny). The CDS then responds with a list of content server locations. Next, the client repeatedly requests groups of chunks which we refer to as chunk windows. For each chunk window, the client sends the server the range of CIDs it wishes to acquire. For our application, we found that using a chunk window of 50 allowed for fewer transactions with the CDS without sacrificing a low rebuffering ratio time. The server then responds with the list of actual CIDs. The client then constructs DAGs using the server information combined with the CIDs, and sends out chunk requests for the entire window. Next, the application goes into an idle mode as it waits for the chunks to be transported and delivered. It does this by repeatedly calling XgetChunkStatuses() (a function included in the XIA API), and waiting for it to return with READY_TO_READ. In the meantime, if the chunk does not exist or is in transit, it returns a vague WAITING_FOR_CHUNK status code.

A key feature to the Video Client is its ability to support seamless failover between different Content Servers. Ideally, the client should just be able to send out its DAG and let the network find a Content Server that has the chunk. In this manner, the application will see no downtime in the midst of content server churn. Thus, a user should not be able to notice changes in the stream, even while in the midst of the most action-packed sequences of Big Buck Bunny.

To accomplish this vision of anycast-like semantics and seamless fallback, the Video Client uses the DAGs in a rather unconventional fashion. Instead of utilizing the fallback paths for scoping and refinement, the DAGs encode alternate content servers as fallback paths. Thus, the client constructs its DAGs to include the CID it wishes to find, as well as several locations where it could expect to find the CID. In turn, these DAGs allow the client to insulate itself from Content Server churn. As long as at least one of the Content Servers is up and running, the network can satisfy the request.

The client also includes application-level failover which, after sending out a chunk request, relies upon continuous WAITING_FOR_CHUNK messages as the signal that the network could not find the content. This mechanism is intrinsically imprecise and would be hard to configure in practice. We offer a discussion of a new return code for XgetChunkStatus() in Section 8.2. The client's response in the face of such an event is an open question. One potential option would be to contact the CID Directory Server and request a new set of fresh content server locations. However, because of the network-level failover, this should only be necessary if the entire current set content servers become unavailable.

4 Implementation

The applications that constitute our network consist entirely of programs written in C++. They are loosely based off of the XIA sample applications.

4.1 Content Server

We constructed our Content Server programs to simply open video files and upload them to their content caches. Following that, they can either terminate immediately without taking down the content, or else wait for some period of time before removing the content. We utilize this flexibility later for testing the scenarios described in Section 5.3. We also attempted to set the TTL of the content chunks, but were unable to find success with the functions provided in XIA's API.

4.2 CID Directory Server

We base the CID Directory Server loosely off of XIA's sample video server. It follows the traditional server model of opening up an XSocket upon which to listen for connections and spinning off a new thread to handle each connection. The type of requests it can handle include requests for the locations of a video (by video

name), requests for a list of CIDs of a particular range, and requests for the videos available for streaming.

4.3 Video Client

The Video Client consists of 2 main components:

1. The Chunk Fetcher Thread
2. The Video Player Thread

Another important, though lesser piece of our architecture is the thread-safe Chunk Queue which serves as the buffer into which we insert chunks prior to playing. We describe each of our components below.

4.3.1 Chunk Fetcher Thread

The Chunk Fetcher thread retrieves the chunks and populates the Chunk Queue with chunks for the Video Player thread to play. It does this by maintaining an Xsocket connection to the CID Directory Server through which it repeatedly requests the CID lists for each chunk window. It then combines those CIDs with the ADs and HIDs of the content server locations, and sends out a chunk request using the XrequestChunks() function from the XIA API. Upon receipt of the chunks, the Chunk Fetcher deposits them into the Chunk Queue.

We support application-level failover by counting the number of WAITING_FOR_CHUNKS messages that are returned in response to the chunk request. The exact value to use for the threshold is unclear – in our testing in a VM with trivial RTTs, four messages was sufficiently high. However, in the context of a real network, the reliance upon the WAITING_FOR_CHUNKS is intrinsically unreliable and overly sensitive to how far away the content might be.

4.3.2 Video Player Thread

The Video Player thread's main task is to remove chunks from the chunk queue and play them to the user. In order to play the video, we incorporate the code base of Plogg, a simple Ogg Theora video player [3], which plays ogg files using SDL and Sydney Audio. The Video Player Thread retrieves chunks from the Chunk Queue and plays the reconstructed stream using Plogg.

4.4 DAG Structure

Figure 2 shows a sample DAG used by our Video Client when requesting a chunk. We construct it in a manner which allows failover to take place starting at any stage of the primary path. For example, if the request reaches the primary host and is unable to find the content, the request will pass to the fallback path and get routed to the second Content Server's AD. If the request fails earlier (i.e. the primary host could not be located), the request is again able to pass to the fallback route. This affords the network sufficient flexibility to find the CID. As long as any one of the content servers is live, the request will succeed and result in the CID getting located and transported back to the Video Client. Note that it is necessary to include fallbacks from each HID, since content can be taken down at any time at a given host, and it will take time before the other entities (and routing tables) get updated to match this altered state.

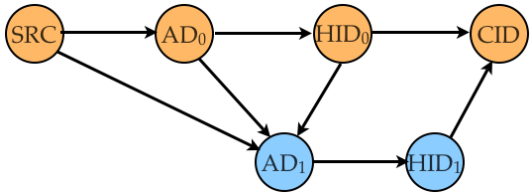

Figure 2: Example DAG. This DAG includes multiple routes: a primary route through the first Content Server and several fallback routes through the second Content Server.

4.5 Network Topology

The XIA prototype comes bundled with a simple network topology that can be used to test simple application scenarios. However, since the network only contains two ADs, each with a single router and host, it was too simple for the tests we wished to run. Therefore, we created the network shown in Figure 3. This network was flexible enough to allow us to host content servers in multiple locations, including in the same AD and neighboring ADs. This network topology follows in the footsteps of the XIA sample topology, as it uses Click to run multiple hosts on a single virtual machine.

5 Evaluation

In order to evaluate the effectiveness of our architecture, we present a number of scenarios which test several failure conditions. All scenarios were tested within the network topology we described in Section 4.5. We omit a benchmark analysis because of the performance limitations imposed by our virtual machine. Many of these scenarios require the starting and stopping of applications at different points in time; we represent all timing numbers in seconds unless otherwise indicated.

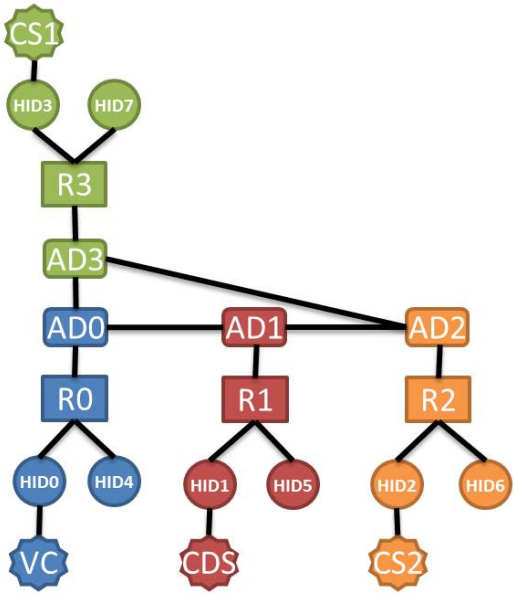

Figure 3: Our custom network topology consisting of four ADs, each with a single router and host. The Video Client is configured to run at HID_0, the CID Directory Server is located on HID_1, and Content Servers 1 and 2 are located on HID_3 and HID_2 respectively

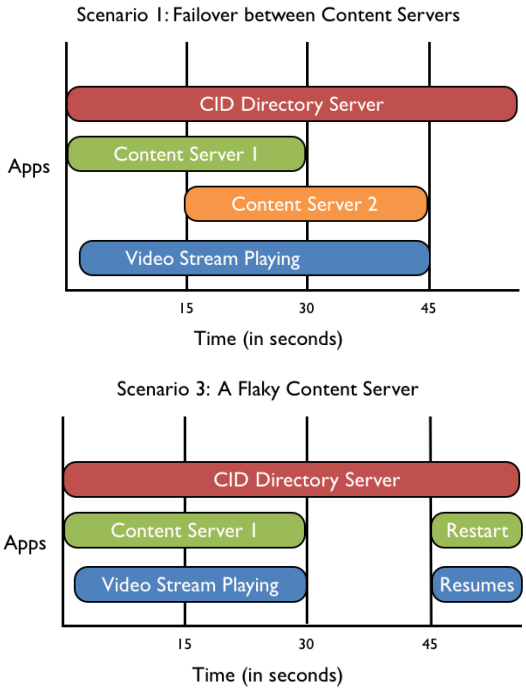

Figure 4: (a): In Scenario 1, two content servers are running, and the client is able to seamlessly continue playing without regard for which content server it is getting the content from. (b): In Scenario 3, the client is contending with the situation of a flaky content server that goes down and then back up again. In this case, the client is able to resume the video from right where it left off.

5.1 Scenario 1: No Content at an HID

Our first scenario is summarized by Figure 4a. In this setup, we launch a Video Client at HID_0, a CID Directory Server at HID_1, and two Content Servers – one at HID_2 and another at HID_3. We next configure the CID Directory Server to provide both locations of the Content Servers to any clients looking for Big Buck Bunny. We configure the Content Server at HID_3 to host the content for 30 seconds and the Content Server at HID_2 to host the content for 30 seconds.

At time 0, we launch the Video Client, the CID Directory Server, and the Content Server at HID_3. The video begins to play, and at time 15, we launch the Content Server at HID_2. At time 30, the first Content Server removes its content, and the network is forced into following the DAG's fallback route to the second Content Server. Finally, at time 60, the second Content Server removes its content, and the video immediately stops.

When running this scenario in our VM, we found that there is no noticeable pause in the video at time 30. The DAG prescribes the fallback formula and the network will seamlessly transition to following the fallback paths. application-level failover never kicks in and is not necessary in this scenario.

5.2 Scenario 2: No Host at an AD

Our second scenario simulates a case where a host goes down by including an invalid HID as the primary route in the DAG. In this setup, we launch a Video Client at HID_0, a CID Directory Server at HID_1, and a Content Server at HID_3. We next configure the CID Directory Server to provide firstly the location of an invalid host (HID_9) and secondly the valid host (HID_3). This will cause all chunk requests of the client's DAGs to first go to the AD of the invalid host, and then, once finding that the host does not exist, the request will head to the fallback path. This scenario does not require any complex timing: at time 0, all applications are simply launched.

When running this scenario on our VM, we found that the video simply starts up immediately without triggering any of the application-level failover mechanisms. Clearly, the DAG used by the client is rich enough to support a scenario where the the primary host is unreachable.

5.3 Scenario 3: Content Down and Back Up Again

We summarize our third scenario in Figure 4b. It attempts to simulate a case of a single flaky content server being responsible for the content. In this setup, we launch a Video Client at HID_0, a CID Directory Server at HID_1, and a Content Server at HID_3 configured to host the content for 30 seconds. At time 30, the content server goes down and the video stops playing. Application failover triggers in the client, but with no result because there are no other content servers up. Next, at time 45, we restart the Content Server at HID_3 and the video smoothly resumes from where it left off.

This scenario demonstrates the advantages of XIA's connectionless chunk request mechanism. Video streaming resumed immediately after its stopping point without any need for connection setup.

6 Discussion

6.1 The Proper Layer of Failover

One of the purposes of this project was to explore how streaming failover might occur in an XIA-based network. As noted in Section 3.3, our video client supports application-level failover not unlike what can be done in IP today. However, given the connectionless nature of chunk requests and the expressiveness of XIA's DAG, the door opens for failover to take place at the network layer. This has many advantages. First, network-level support for failover has the opportunity to effectively eliminate client downtimes in the face of changing content server dynamics. If failover is performed at the application layer, clients may experience an insignificant amount of downtime as they wait for timeout to occur and for a new connection to be set up to a new backup server. This set of delays can be especially damaging to traffic like video, which is sensitive to jitter. Second, it reduces application-level complexity. Rather than requiring clients to include highly-optimized logic to support failover on the fly, XIA has the opportunity to abstract away this complexity.

While network-level failover does violate the End-to-End Argument [7], we note that a failover mechanism can apply to other principal types such as SIDs. Hence, such functionality has the opportunity to improve performance and reduce complexity for a vast number of use cases.

6.2 Limitations of the DAG for Content Failover

Given today's implementation of XIA's DAG requests, we admit that there are some severe limitations with our approach. Failover as we have proposed it would not be practical in the context of a real application due to the limited number of fallback paths allowed in a DAG. Since XIA limits DAGs to having only four fallback paths [4], a video client is only able to include at most two content servers in a DAG. While this effectively allows an application to specify a single backup server, this would fall short of the flexibility and power of anycast-like semantics. Allowing DAGs to specify a greater number of fallback paths would open the door for developers to harness the full potential of network-level failover. The four-fallback path limitation is also malevolent to our approach in that it disallows our video client's DAGs from being able to take advantage of scoping and refinement.

7 Challenges

In addition to our stated project goal of creating a streaming video application specific to XIA, we also were in the unique position of being among the first XIA small developer teams without any prior XIA experience. Our experience creating a network application while navigating the XIA network configurations and beginner's wiki guide allow us to present our experience to the XIA research community and offer recommendations to assist future developers. The following sections describe the challenges of developing for Version 1.0 of the XIA Prototype.

7.1 Network Setup

While the XIA Prototype's sample topology is simple to establish and run, we ran into great difficulty when implementing our own custom network topology. Although it might seem like a trivial task, we ended up having to invest a considerable amount of time and effort into troubleshooting our topology.

To begin with, the XIA requires xsockconf.ini files that link a application with a specific Click port. These files are absolutely necessary for directing the network traffic to the appropriate host. The wiki pages specify that the network reads these files to establish the basic services. However at the time of writing, the wiki still has a TODO for telling developers where to place these files. After many frustrating days, we eventually sent an email to the XIA help desk, who politely informed us that XIA requires the xsockconf.ini files to be located in the directory from which the executables are launched.

It was an easy fix, but not one that is altogether intuitive for someone unacquainted with the inner workings of the XIA network daemons.

In addition, other files spread throughout the prototype needed alterations to allow XIA to function. /etc/hosts.xia stores the RE addresses for the various hosts and routers but does not warrant a mention in the wiki. Another problem arose when we tried to create more than six hosts. This change led to a storm of initialization errors with little useful error text. Eventually, after several hours of debugging, we discovered the cause of the problem: the file /etc/click/xia_address.click contains 6 hard-coded HID addresses. Adding more hard-coded HIDs to this file fixed the problem. To our knowledge, the xia_address.click file was not mentioned anywhere in the wiki.

Ultimately, we created our own xsockconf directory which includes our Click topology files and launching scripts. Central to its success is the makefile, which carries out many of the duties done by the xianet script in the sample topology. It distributes configuration files to all of the relevant directories, and then starts up the various daemons that run the network.

7.2 Playing Video

We found that playing video at a low-level within the context of a C++ program is not trivial. The realm of video playback is vast with a myriad number of video formats to choose from, and few example applications. At first, we naively thought that we could just follow some tutorials and get a video to play within a few dozen lines of code. However, we soon discovered that any foray into video playing would require extensive use of third party libraries and platform-specific technologies. Fortunately, we discovered Plogg, a relatively simple video player that uses the Ogg Theora Library in conjunction with the Sydney Audio library and Simple DirectMedia Layer (SDL) to decode and play an ogg video file. Plogg, however, has not been updated in over two years, and as such, has fallen behind the updates of other libraries. To get it working, we needed to roll SydneyAudio back a few revisions. Also, inexplicably we could not get video playing to work on one of our VM's (the client was able to play only the audio).

7.3 XIA API Change

In mid-march, the XIA 1.0 API was released, which included several changes to the fundamental XSocket functions. To cope with this, we needed to spend a large amount of time adapting our code to work with the latest API changes. Luckily, the XIA update also included updates to some of the XIA sample applications, and we were able to fashion our alterations after those.

7.4 XIA Update Invalidated Our Approach

On April 9, an update to XIA's source snapshot was pushed to the XIA github repository, which included the following commit message: "Fixed intra-ad routing between hosts". We found that this update invalidates our approach to using DAGs, as it prevents all of the network-level failover scenarios from successfully working. When running our applications in two VM's - one with the version of XIA released on April 5, and one with the version of XIA released on April 11, we find that our architecture only works with the old version of XIA.

7.5 Bug in Sample XIA Video Server Application

We were slowed by the presence of an existing bug in the sample XIA video server application. As forewarned by the comment at the top of the sample program, "seems to have some issue currently ... for multiple clients", the bug had to do with multi-threading. In case this bug has not yet been discovered by the author of video_server.c, it occurs at line 282, where a local variable for acceptSock is passed into pthread_create() instead of a dynamically-allocated integer. Each request-handling thread was referring to the same socket descriptor.

8 Recommendations for XIA

In this section, we present a number of suggestions for XIA, from a developer perspective.

8.1 Network Package instead of Script

As mentioned in Section 7.1, we encountered a great deal of difficulty in creating our own custom Click-based topology. We eventually were able to create our own xsockconf bundle through heavy reliance on examining existing start scripts and performing trial and error. Clearly such challenges represent an incredible burden on any beginning XIA developer, and we feel that the process should be streamlined. In order to accomplish this, we recommend that a basic network package file (similar to our xsock config directory) be included as part of XIA's standard files. Such a package would be of tremendous help to beginning developers with little Click experience and would drastically lower the activation energy for folks who are interested in writing their own XIA applications.

8.2 Additional Request Chunk Response Code

In Section 4.3.1, we mentioned the difficulties faced when implementing application-level failover. In particular, we found ourselves restricted by the limited set of response codes that can be returned by XgetChunkStatuses(). There are currently four response codes in today's XIA API:

1. **READY_TO_READ if the requested chunk is ready to be read.**
2. **INVALID_HASH if the CID hash does not match the content payload.**
3. **WAITING_FOR_CHUNK if the requested chunk is still in transit.**
4. **REQUEST_FAILED if the specified chunk has not been requested**

The problem lies in identifying a request that fails when it gets to a host that does not have the CID we are looking for. Currently, XgetChunkStatuses() will handle such an event by repeatedly returning back WAITING_FOR_CHUNK, which is indistinguishable from the output indicating a chunk in transit. We were forced to rely upon the receipt of multiple WAITING_FOR_CHUNK messages as a signal to indicate that application-level failover should occur. It is unclear how an application developer could set such a threshold with much confidence. Given this, we propose that a new return code be added to XrequestChunks():

- **CHUNK_NOT_FOUND if the specified chunk does not exist at the location**

This new status code would be of incredible use to any application developer in need of supporting application-level content probing.

8.3 Increase the Number of Fallback Paths Allowed in a DAG

As discussed in Section 6.2, our approach is severely limited by the four-fallback-path limit imposed by XIA's DAG specification. Backup-server failover semantics are workable in the current limitation space, but would require no fewer than eight fallback paths in order to support scoping and refinement. If anycast-like failover semantics were desired, this fallback-path limit would have to be substantially increased.

9 Limitations & Future Work

9.1 Port to a Physical Network

Due to the nature of our video architecture residing within a virtual machine with limited resources, the performance of our video stream was never able to play a video without a large amount of buffering. This was due to the limited resources allotted to our virtual machine, which was unable to handle the stress of the many network daemons needed to run the Click topology. Thus, to assess the performance of our applications and techniques, testing would need to take place on a physical network.

The relocation of applications to a physical network would also open the door to testing true failure scenarios as well as client migration.

9.2 A Content Server-Aware CDS

Our current implementation of the CID Directory Server merely includes hard-coded ADs and HIDs for the various Content Servers. Obviously, such a technique is not workable in a real architecture and would need to be replaced by some form of content server registration. Content Servers could be configured to contact the CID Directory Server periodically in order to establish liveliness. This would allow the CID Directory Server to send fresh lists of Content Servers to the clients based on its current network knowledge.

Also, it is unclear how a CID Directory Server might determine which set of Content Server locations to send back to a connecting host. It could do this by choosing Content Servers on the basis of fewest hops, geographic location, or some means of client information gathering akin to that done by the Video Control Plane architecture [5].

9.3 Video Client User Experience

The current version of the Video Client is very barebones, and would not be suitable for distribution to an actual user base. To approach beta-quality functionality, one would need to introduce basic user services such as video selection and a GUI.

9.4 Source and Demo Video

A github repository for our source can be found at
https://github.com/StephenBrownCS/
xia-video-streaming

A tar archive of our source files can be found at
http://cs.wisc.edu/~sbrown/downloads/

xia-video-streaming.tar

A video demonstrating scenarios 1 and 3 can be found at http://cs.wisc.edu/~sbrown/sharing_big_buck_bunny_in_xia.html

A tar archive of this report can be found at http://cs.wisc.edu/~sbrown/downloads/xia-video-streaming-report.tar

10 Conclusion

In the last thirty years, the primary use of the Internet has evolved from Telnet, email, and FTP traffic to web and video centric services. However, the protocol constraining the usefulness of that traffic has changed only at a glacial speed. In response, service developers have had to tunnel, encapsulate, and perform other tricks to squeeze as much performance as possible out of the current Internet architecture.

In an attempt to improve the infrastructure of the Internet, not just for today but for the foreseeable future, researchers at UW-Madison and CMU developed XIA. The architecture is perfectly suited to support today's content-oriented applications by allowing the network to natively resolve the client's intent. It is within that framework that we set out to develop a streaming video application that incorporates XIA's strengths.

To accomplish this, we created a video application architecture which leverages the connectionless nature of XIA's chunk requests, and the expressiveness of its DAG addressing. Our architecture supports failover at both the network and application-levels, and offers interesting insights into the problems XIA application developers face. Based on our experiences, we were able to make several recommendations toward improving XIA.

We believe that XIA has opened exciting new opportunities for network application developers. It was our primary goal to create a working streaming video application, but we hope that our work will help carve a path for future application developers who wish to explore XIA.

References

[1] Chris Anderson and Michael Wolff. The Web is Dead. Long Live the Internet. http://www.wired.com/magazine/2010/08/ff_webrip/all/, August 2010.

[2] David Clark. The Design Philosophy of the DARPA Internet Protocols. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 106–114. ACM, 1988.

[3] Chris Double. Plogg. https://github.com/doublec/plogg, February 2010.

[4] D. Han et al. XIA: Effcient Support for Evolvable Internetworking. *Proc. 9th USENIX NSDI*, 2012.

[5] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 359–370. ACM, 2012.

[6] Yves Poppe. IPv6: A 2012 Report Card. http://www.circleid.com/posts/20121128_ipv6_a_2012_report_card/, November 2012.

[7] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[8] Doug Webster. The Cisco Visual Networking (VNI) Forecast 2009-2014. http://blogs.cisco.com/news/cisco_vni_forecast_2009-2014/, June 2010.