



Georg-August-Universität
Göttingen
Institut für Informatik

ISSN 1611-1044
Nummer IFI-TB-2007-03

Technischer Bericht

An Experimental Analysis of Joost Peer-to-Peer VoD Service

Jun Lei, Lei Shi and Xiaoming Fu

Technische Berichte
des Instituts für Informatik
an der Georg-August-Universität Göttingen

15. October 2007

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel.	+49 (5 51) 39-1 44 14
Fax	+49 (5 51) 39-1 44 15
Email	office@informatik.uni-goettingen.de
WWW	www.informatik.uni-goettingen.de

An Experimental Analysis of Joost Peer-to-Peer VoD Service

Jun Lei, Lei Shi and Xiaoming Fu
 Computer Networks Group, University of Göttingen, Germany
 {lei,shi,fu}@cs.uni-goettingen.de

Abstract—Most of the current Video-on-Demand (VoD) systems rely on content distribution networks or some local streaming proxies. While these traditional systems offer a means for media delivery and streaming, they also pose a significant performance challenge in terms of scalability and service delay as the number of clients increases. To solve this issue, peer-to-peer (P2P) technologies have been applied to support the VoD systems. Joost is one of such systems for distributing TV shows or other forms of video over the Internet. However, like Skype in its early stage, the mechanisms behind Joost are still unrevealed.

The main purpose of this paper is therefore to study the underlying Joost architecture and its key components, and analyze its media streaming behaviors and peer management mechanisms through close investigations on Joost network traffic. With three envisioned typical scenarios we have further studied the Joost performance in terms of locality awareness, bandwidth capacity and VoD functionalities. Based on extensive experiments, we infer that Joost is a server-assisted peer-to-peer VoD system. It mainly relies on a set of delicate infrastructure nodes (e.g. content servers) for video distribution. To our best knowledge, this paper is the first comprehensive analytical and performance study on commercial P2P VoD services.

Index Terms—Peer-to-peer (p2p), Video-on-demand (VoD), Performance measurement

I. INTRODUCTION

In the recent few years, IPTV has gained a tremendous popularity in the operators and users as well as a lot of attention from the research community. For residential users, IPTV is often provided in conjunction with Video-on-Demand (VoD) and may be bundled with Internet services such as VoIP. Traditionally, when a client user selects a program, a point-to-point unicast connection is established between his (or her) decoder (aka *set top box*) and delivering media server, which lacks efficiency and scalability. Most of the current VoD services mainly rely on content distribution networks (CDNs) [20] or some local streaming proxies to increase system scalability and to alleviate the delay experienced by end users. However, their system performance and deployment become a key challenge as the number of clients increases. Especially, if a flash crowd [15] occurs, servers can be easily overloaded. The similar phenomenon occurs when a web site catches the attention of a large number of people, and gets an unexpected volume and possibly overloading surge of traffic. To address this issue, peer-to-peer technologies have been employed to support VoD services. Joost [1], created by N. Zennström and J. Friis, co-founders of Skype [29] and Kazaa [23], is one of such systems for distributing TV shows or other forms of video over Internet using P2P TV technologies.

We choose Joost as the target for the study of Peer-to-peer VoD services due to the following reasons. Firstly, as one of the earliest and best-known commercial peer-to-peer VoD products, Joost has the potential to become popular following a successful story of Skype. It offers high-quality and comprehensive VoD services, for instance, the current version (Beta 1.0) supports an instant on-demand video without any need for additional set top box. Furthermore, it is provided as a freeware without releasing source code, although it is known to be built on top of several open software such as Mozilla/xulrunner [21]. These facts may provide us some means to understand some particular behaviors of Joost clients, however, except for the limited knowledge of the used open software, the underlying P2P architecture and detailed mechanisms/techniques used in Joost, like when Skype was new, are still unrevealed. Getting deep insights into various aspects of Joost has been challenging because the Joost architecture and many technologies it uses are proprietary. In particular, in order to understand its performance, we had to collect a large amount of data and analyze media streaming behaviors and peer management behaviors.

In this paper, we first raise some questions about the Joost system and seek to answer them through numerous experiments:

- *What is the Joost architecture?* What are the key components in such an architecture? Which functions are performed by these components? How these functions can be achieved?
- *How are the characteristics of Joost network traffic?* Which kind of protocols does Joost use? What is the fraction of outgoing and incoming data traffic? What is the fraction of network traffic that a peer receives is control traffic?
- *What are the characteristics of peer behaviors?* At what rates does a peer download from and upload to its partners? How are the partnerships different for a University LAN client and a DSL residential client?
- *How are the Peer-to-Peer technologies used in Joost?* How does the peer selection performed in Joost? During the peer selection, has Joost considered locality? Whether heterogeneity is considered? How about the fairness of contribution? Has Joost considered peer adaption during dynamic changes?
- *How does Joost provide the VoD services?* Which kind of the VoD functions provided? What is the media streaming

behavior? How does the peer management involve in these services?

- *How about the performance with different network conditions?* How about the performance of high capacity nodes, if they can offer high network access speed? If the Joost client suffers from low, unstable network conditions, will the performance dramatically degrade?

The rest of the paper is organized as follows. In Section II we give a brief overview of Joost software. Section III describes the key components of Joost system. Section IV discusses key Joost functions like installation, bootstrapping, reconnection, channel switching and VoD functionalities. Based on above studies, we infer the Joost architecture in Section V. As the performance aspect plays an important role in P2P VoD user adoption, we envision three typical usage scenarios and use them to study more closely the behaviors and performance of locality awareness, bandwidth capacity and peer management in Section VI. In Section VII we review related work. Finally, we conclude this paper and plan future work in Section VIII.

II. JOOST OVERVIEW

The current Joost Beta version (Beta 1.0) runs on Windows XP, Windows Vista and MAC OS X. It supports 15,000 TV shows through 250 channels [1].

After registration, each client can log into the Joost system directly. A Joost client user can select from the channel list which specific program to watch. If the program is completely new to the client, it may take up to 20 seconds to really start the program. Most likely, the time is required for requesting contents from other peers and preparing downloading. Otherwise, the program will start immediately after selection since content can be directly fetched from the local video cache seen in Section III.

Moreover, the Joost client can browse the channel list and add selected channels into “My Channel” list that is a favorite channel list for client’s convenience. It usually takes 7-9 seconds to switching the channels.

III. KEY COMPONENTS OF JOOST SOFTWARE

According to our following experiments, during its operation a Joost client performs one or more of the following actions: listen on particular ports for incoming traffic; store media data into its local cache; maintain a table of other peers called a host cache; use Advanced Video Codec (AVC); determine if it is behind a NAT or firewall; and functions required by additional features, such as instant messaging. This section discusses the key components involved in these actions.

A. Ports

During the installation and bootstrapping, Joost client contacts some HTTP/HTTPS servers initially. It will be further described in Section IV.

Upon the first initialization, the Joost client (JC) randomly chooses a port number through which the JC can subsequently

communicate with other peers and Joost servers. Such a port (noted as JC_P) is usually some high port (e.g. 57929). Once the port number is determined during the first run, subsequent media transactions will always use this port no matter the JC restarts or reboots. Besides, the Joost client listens on this port for incoming requests from other peers. To send media data to other Joost clients, the JC also uses this port.

In April 2007, port number 4166 was assigned by the Internet Assigned Numbers Authority (IANA) [16] as the official TCP and UDP port used for Joost. Since then, all media data and some of the control messages (e.g. peer management) are sent through 4166 from Joost servers (seen Section V for detailed information). Looking into the specific port number facilitates our following experiments.

To summarize the above analysis, the different ports used in Joost network traffic are depicted in Table 1.

TABLE I: Ports

Protocol	Joost Server	Joost Client	Super Node
HTTP	80	HTTP port	
HTTPS	443	HTTPS port	
TCP	4166	JC_P	4166
UDP	4166	JC_P	4166

Here, *super node* is not a Joost Client, but a delicately deployed entity mainly for peer management and peer lookup purposes in Joost, as described in Section V.

B. Video Codecs

Joost claims to use “an H.264 codec for video encodings (aka AVC, aka MPEG-4 Part 10, aka ISO/IEC 14496-10) called CoreAVC, created by CoreCodec” [22]. However, in our experiments we did not observe H.264 as shown in Figure 1. We conjecture that RTP dynamic (payload type: 96-127) is the codec H.264 (payload type: 99) for Joost video encoding as the freely available analyzers we found were unable to distinguish it. Furthermore, it was observed that Joost used G.711, G.726, G.728, G.729, G.723.1 and GSM for audio codecs (Figure 1), which allow frequencies between 8,000-90,000 Hz to pass through. These codecs have been developed by ITU-T [17]. We conjecture that Joost followed the RTP specification [31] for the implementations.

C. Local Video Cache

A JC for Windows XP users stores the media data in its local cache as “anthill_cache” at System_Disk(e.g. C:)\Documents and Settings\

The cache size depends on which and how long programs have been played. Each time a new program is chosen, the size

Protocol	Percentage	Bytes
UDP	89,862%	2.648.613.744
RTP Dynamic	6,282%	185.161.606
HTTPS	0,707%	20.849.981
RTCP	0,491%	14.481.126
G.711	0,430%	12.662.605
IMAP	0,218%	6.415.720
G.729	0,202%	5.939.730
G.726	0,198%	5.826.491
G.723.1	0,197%	5.816.021
RTP	0,196%	5.774.642
G.728	0,196%	5.768.628
H.263	0,195%	5.741.342
H.261	0,195%	5.735.162
GSM	0,194%	5.721.785
ARP Request	0,140%	4.134.996
SAP Request	0,063%	1.848.128
ICMP Dest...	0,049%	1.447.486
TCP	0,042%	1.251.630
HTTP	0,041%	1.194.632

Fig. 1: Example of protocols used in Joost system.

of the cache will automatically increase. In our experiment, it was more than 2 GB. Therefore, we believe that user’s system resources will be significantly occupied if the JC continues to watch different channels.

If we assume that the local cache did completely store the played video, the JC should watch the old program directly from the local cache. However, when we had disabled the Internet connection, the program surprisingly stopped, even if the particular program has been watched 1 minute ago. We guess that although some media data have been stored locally, it still requires a kind of codec from the remote server or a encryption key (e.g. AES key) authorized by the Joost server to access the video file. To prove these conjectures, we made additional experiments.

We launched a new channel and at that moment, the local cache was empty. After the whole channel was watched, the size of cache file grew up to 1.7 GB and the average download speed was 518 kbps. If we turned off the JC and restarted it, the download speed was dramatically dropped down to 11 kbps when the same channel was watched. Moreover, the size of local cache increased only 5.88% during the second watching time.

D. Host Cache

Similar to what was observed in the Skype analysis [29] and [30], host cache is a list of Joost super nodes IP address and port pairs that JC builds and refreshes periodically. The JC for Windows XP stores the host cache as an XML file “shared.xml” in System_Disk:\\Documents and Settings\\<XP User>\\Application Data\\Joost\\anthill. A Joost client for Windows Vista stores it in System_Disk:\\users\\<Vista User>\\AppData\\Roaming\\Joost\\anthill.

E. NAT and Firewall

We detected that a random port was configured at the first login time and kept in use for the subsequent media transmission. As video packets are sent over UDP (as shown in Section V), we conjecture that Joost uses a modified STUN [19] protocol to determine the type of firewall and NAT it may

be behind, similar to what was observed in [29]. The NAT and firewall traversal related information is stored in the share.xml file.

More information related with STUN is stated in Appendix.

F. Additional Features

Joost utilizes widgets to provide some additional functionalities, among which the most notably one is a channel-based chat room. Based on this feature, clients are able to talk to each other in real-time when watching the same channel.

Furthermore, it is possible for clients to create a private channel by dragging the video to the show bar.

Since during our experiments there was only a limited number of Joost users, it was hard to experience a comprehensive set of these additional features. Thus, the rest of this paper focuses on the primary features as described in previous sections.

IV. JOOST FUNCTIONS

All the experiments were performed for Joost version Beta 1.0. Joost was installed on Windows XP and Windows Vista machines. The Windows XP was Intel Pentium Dual-Core 1.73 GHz processor with 1.00 GB RAM. The Windows Vista was equipped with AMD Althon X64 processor with 1.00 GB RAM.

A. Installation

One Joost server was involved in installation phase: lux-backend-lo-1.joost.net (89.251.4.75). The client sent a HTTP 1.1 GET request to this Joost server and downloaded a SQLite [28] file (zelos2.sqlite) which is the initial channel list. See Appendix for complete messages.

This channel list is stored in System_Disk:\\Program files\\Joost\\defaults\\profile\\zelos2.sqlite, currently fixed to 1.35 MB size and 33 channels). Clearly, SQLite is used for the Joost channel database management, which is a self-contained, embeddable, zero configuration SQL database engine. Figure 20 shows a snapshot of the initial Joost channel list.

At that moment, the local cache, node identity and the listening port number through which the client will communicate with other peers were not yet configured. We found that there was no local cache file, no share.xml file in which node identity and port would be configured. In this paper, we use the term of peer and client interchangeably.

B. Bootstrapping

Totally, three Joost servers and two Joost super nodes were responsible for the bootstrapping procedure, by which the listening port was configured and the channel list was updated (System_Disk:\\ Documents and Settings\\<XP User>\\ Application Data\\Joost\\Profiles*.default\\zelos2.sqlite, 1.76 MB, 45 channels).

Firstly, the JC communicated with lux-www-lo-2.joost.net server (89.251.2.85) over HTTPS. We conjecture that it is a kind of tracker server. In case the newcomer contacts the tracker, it will receive some available super node addresses

and possibly some content server addresses. Then, a HTTP GET request was sent to lux-www-lo4.joost.net (89.251.2.87) server for getting the latest software version. See the Appendix for the detailed message exchanges.

Besides, the lux-backend-lo-1.joost.net server (the same server involved in the installation) was also involved in bootstrapping the client by sending packets over HTTPS.

Finally, JC started to contact some of Joost super nodes, for instance, lid-snode-1-eth0.joost.net (89.251.0.16), lid-snode-2-eth0.joost.net (89.251.0.17) and lux-snode-1-bond0.joost.net (89.251.4.71), possibly to obtain the list of other available clients and begin transacting video contents. Before long, the running JC has already started communicating with other peers besides Joost servers.

Figure 2 shows the throughput of above three Joost servers during bootstrapping. At the very beginning, the tracker server (lux-www-lo-2.joost.net) helped bootstrapping the new client. Clearly, after a short period the tracker server was not involved in the subsequent communication. Then, the backend server (lux-backend-lo-1.joost.net) appeared and continuously sent a large amount of data to the client, we guess, in order to update the channel list. At some point of the bootstrapping procedure, the version server (lux-www-lo4.joost.com) checked the version of Joost software.

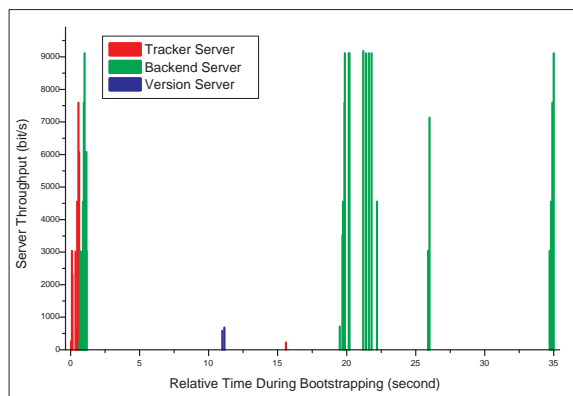


Fig. 2: Joost server throughput during bootstrapping.

C. Reconnection

We restarted the JC and attempted to observe the peer behaviors during client reconnection. The above-mentioned initialization process occurred again with two exceptions. One is that the version server might not appear if the interval was short and the other exception is the port number that was negotiated when first connecting to the Joost network was stored in the share.xml file and reused. If the version server really appeared, checking software version used HTTPS instead of HTTP.

Furthermore, the JC attempted to communicate with peers from which it has downloaded content previously. This was done by sending some small UDP probe (64 bytes) to other clients, which in turn would reply with another small UDP probe (64 bytes). Afterwards, media data was continuously downloaded from some connected clients.

D. Channel Switching

During our experiments, Joost classified the channels into 13 categories: Explore, My Channels, What’s popular, Cartoons & Animation, Comedy, Documentary, Drama, Entertainment, Film, Lifestyle, Music, News, and Sports & Games. All the selected and viewed channels are kept in the “My Channels”. Thus, there are two possibilities to switch the channels. If there are no selected channels in “My Channels”, the client needs to firstly browse the channels and then choose one. Otherwise, the client can just pick one channel from the “My Channels”. The second option is much faster than the former one because channel browsing takes a long time.

1) *Channel Browsing*: Since the initial channel list was downloaded from the lux-backend-lo-1.joost.net, we conjecture that this server is also responsible for the channel management, such as channel list downloading and updating.

We traced the network utilization of the backend server when the client browsed the channels. In Figure 3, the column represents the network utilization (in bit/s) during the channel browsing. Between time 40 – 60 seconds there was a large amount of the traffic and at exactly that time, we browsed the channels. If we selected a certain category and stopped browsing, the utilization dramatically dropped and was kept stable. Figure 4 depicts that compared with the channel browsing the network utilization between 60 and 120 seconds was relatively low. This indicates that the channel list was dynamically downloaded from the server.

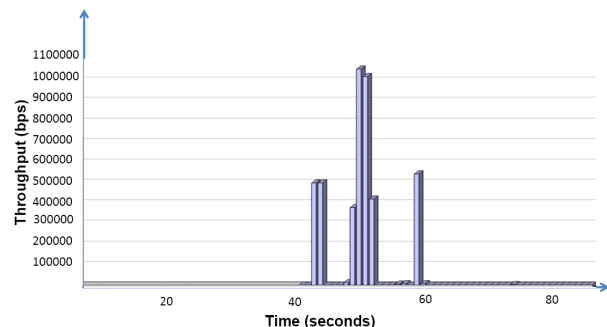


Fig. 3: Browsing Channels.

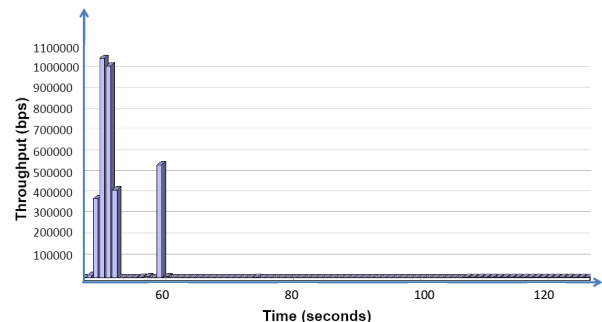


Fig. 4: Stop Browsing in Certain Category.

2) *Switching Channels*: We observed that at the moment of channel switching the JC contacted firstly with some super

nodes. These super nodes IP addresses and ports have been obtained from the tracker server (lux-www-lo-2.joost.net) during the initialization. For example, the JC contacted lid-snode-2-eth0.joost.net (89.251.0.17) and lux-snode-1-bond0.joost.net (89.251.4.71) over UDP. From these super nodes, the JC may obtain some address lists including related content servers and possibly some other clients who were watching the same channel but ahead of this particular client. Once the JC received such a list, it attempted to contact them by immediately sending UDP requests. At the same time, other super nodes continued to send the client available address lists. When the selected channel started playing, the JC periodically exchanged messages with these super nodes over UDP. We believe that the super nodes are responsible for redirecting clients to content servers or peers during channel switching. Moreover, they periodically exchange messages with clients, possible for the purpose of peer management or acquiring keying materials to eventually watch the video stream.

In the current version of Joost, the function of local video buffer is not supported. That is, when the client pauses the video it stops downloading. There are some claims that Joost should have a small amount of buffer in order to avoid the stuttering and temporary freezes [22]. However, observing from the fact that most of users frequently switch channels, the current solution may save resources in case of short-term switching as it does not maintain local buffers.

E. VoD Functionalities

Unlike file sharing or live media streaming, each JC is more “selfish” in the sense that it only cares about contents after its current playing position, which is often different from other peers. The peer can only download from those whose playback positions are ahead, or from who have already watched the program. Instead, itself can help peers which join later. However, as each Joost client can change its playback position at any time, which differs from many other P2P streaming systems, it becomes difficult to optimize the overall VoD system. For example, the “rarest-first” strategy [10] in BitTorrent is not applicable here.

As a result, the VoD aspect attracted our particular interests. After repeating several experiments, we come up with the following conclusions.

First, in Joost system each media file was broken down into fixed-time chunks and each chunk is encrypted. During our experiments, if the fast forward interval was smaller than 5 seconds the JC may continuously play without waiting. However, if the interval is large it took 5 – 10 seconds to start playing. To illustrate our observation, we suppose that each media file is divided into multiple 10-second play time chunks, but the exact size of the chunk is unknown. As shown in Figure 5, each chunk includes an anchor which is a dedicated marker for encrypted media data similar to I-frame in MPEG [9]. When a seek is triggered in a client (i.e., control bar is moved to a backward position), the client will always search for the closest anchor in the local video cache if it is already downloaded. Otherwise, it firstly sets a new anchor and requests new data from other peers.

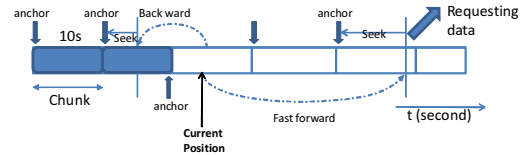


Fig. 5: On-demand Video Functions.

Second, if the JC drags the control bar into any specific position, it communicates with one of the super nodes, for example, lid-snode-2-eth0.joost.net (89.251.0.17) or lid-snode-1-eth0.joost.net (89.251.0.16) in all our experiments.

To prove that those super nodes support VoD functionalities, we traced the first super node during the periodic (every 20 seconds) actions of “fast forward” (10-minute period of video) within the same program. As shown in Figure 6, each time the JC dragged the control bar, there was a large amount of traffic sent from the super node. Otherwise, the traffic from the super node was quite low compared to the “fast forward period”. By analyzing the traced data, we found that UDP was used to carry the traffic and the average received packet size was 137 bytes and the average size of sent packets was 141 bytes (all below 150 bytes). Therefore, we suppose that these packets are only used for control, not for media transmission. Furthermore, we conjecture that the updated lists, which contains information about peers having already received the on-demand contents, are encoded in these packets.

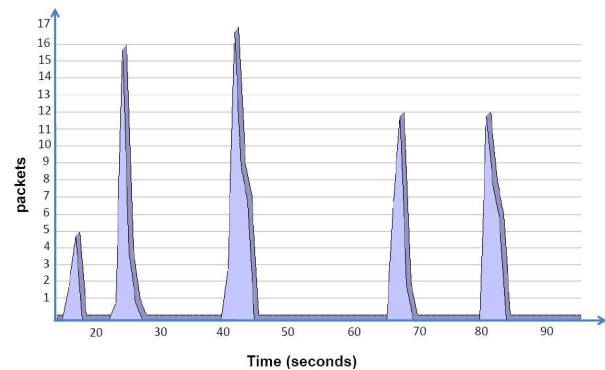


Fig. 6: VoD functionality

V. JOOST ARCHITECTURE

Based on the above observations, we deduced the basic architecture of the Joost system. As depicted in Figure 7, there are five different types of servers, Joost super nodes and Joost clients. Obviously, there are other servers taking charge of added value services, for example, instant chat service. Because the fundamental functions are our focus, these additional servers have been removed from the figure.

In the following section, the server architecture, Joost super nodes and protocols used in the Joost network traffic will be described in detail.

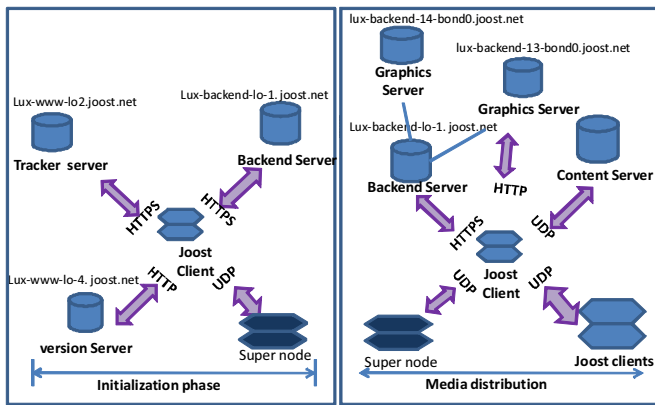


Fig. 7: Joost Architecture.

A. Server Architecture

Five types of servers are participating in the Joost architecture as seen in Figure 9, respectively, version server, tracker server, backend server, channel graphics server and content server.

The lux-www-lo4.joost.net server is the version server that is responsible for checking the current version of the software. When the JC is crashed, this server is also responsible for the error reports.

During the initialization, lux-www-lo2.joost.net takes charge of sending the initial peer list that includes some of the super nodes and content servers. After that, the tracker server will not appear in any of the other stages (e.g. channel browsing, switching). Such a server is not responsible for browsing program, nor for channel switching. Its only job is to keep track of its membership and helps bootstrapping new peers. In the later stage, this server does not appear since peer communication can continue without the tracker.

The client sends HTTPS request to lux-backend-lo-1.joost.net, the backend server, that performs channel list management (e.g. updating, downloading) and load balancing. Switching channel will cause significant traffic, which can be observed in Section IV. Besides, it periodically (every one minute, 81 kb traffic) communicates with the client.

The detailed message flow is shown in Figure 8.

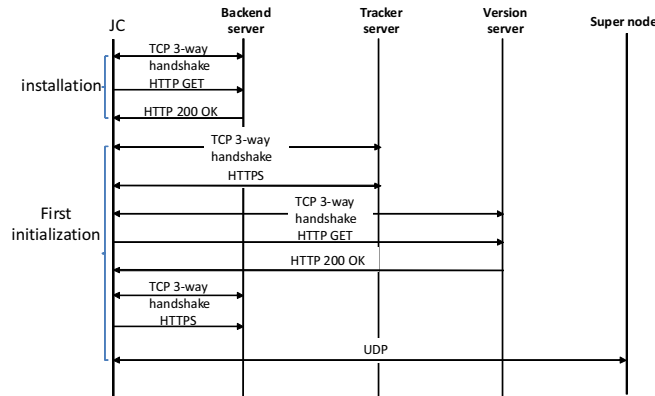


Fig. 8: Installation and First Initialization Message Flow.

Later, we realized that the channel management is actually not performed by a single server, but a server cluster. That is, the lux-backend-lo-1.joost.net server answers for controlling channel requests and keeping load balance among cluster servers, whereas other cluster servers are only responsible for a certain task (e.g. channel graphs downloading). See Appendix for the complete messages. For instance, there was a server, lux-backend-13-bond0.joost.net (4.251.4.153) from which the JC downloaded the channel graphs instead of from the main control server.

Especially, when the control server is heavily overloaded, some of the channel list updating and channel graph downloading actions will be taken over by other cluster servers. Another task of the backend server is providing searching services for channels or specific contents.

The last type of Joost server is the content server. Joost did distributedly deploy a serious number of content servers over the network. During our experiments, we observed the following server sites:

- 4.71.105.0/24 (sna-Itsnode- x -bond x - x .joost.net)
- 4.71.174.0/24 (IPsoft)
- 212.187.185.0/24 (Icy-Itsnode- x -bond x - x .joost.net)

Here, x varies from 0 to 10. The first and third IP address site is owned by Level 3 Communication INC [24] which has been selected by Joost to support on demand Internet TV. Since July 24, 2007, Level 3 provides Joost with network solutions including high speed Internet access and co-location services in North America and Europe [24]. The second IP space group belongs to IPsoft service provider in New York.

B. Super Node

Different from other P2P networks (e.g. Skype) or overlay multicast solutions [7], these nodes are only used for controlling and helping new peers find contributing peers. They are not responsible for relaying/forwarding media data to other peers. It is quite efficient and reasonable since peer management is split from the media distribution, which not only eases the management but also improves the efficiency of transmission. Differently, if a super node in Skype leaves ungracefully, all the other peers relying on it will be unavoidably affected.

To summarize, Joost super nodes perform the following three basic functions in most cases.

- After bootstrapping JCs first contact super node, which directs clients to available peers. Peers are either JCs or Joost content servers.
- For on-demand video functions, super nodes periodically exchange some small UDP packets with clients. We believe that these UDP packets are used for peer management, such as keep-alive probing.
- Additionally, channel switching requires the JC to talk to the super node. At that time, super node most likely helps it finding available peers to fetch the new media data.

C. Protocols

Figure 9 depicts the main protocols used in the Joost system.

Protocol	Functionality	Packet Size
UDP	Video distribution	1104 Bytes
	Content Probe (peer to peer)	~ 64 bytes
	Channel Switching (peer <-> super node)	< 1000 bytes
HTTPS	Administrative management	
	Client -> Server	64 bytes
	Server -> Client	<=1518 bytes
HTTP	Software version	
	Client -> Server	~64 bytes
	Server -> Client	< 500bytes
	Channel management	
	Client -> Server	~ 64 bytes
	Server -> Client	<= 1518 bytes

Fig. 9: Main protocols used in the Joost system.

As shown in Figure 9, all video packets are encoded in UDP and the size is exactly 1104 bytes. It is observed that JC uses the negotiated port number (JC_P) with other clients. Besides the media transmission, peers frequently negotiated each other by sending UDP probes (64 bytes). During the channel switching, peers contact the super nodes by exchanging UDP packets.

There are occasionally some TCP packets sent between the super node and the JC. We suppose that these TCP packets are only used to probe whether the JC is still alive. Furthermore, we tracked the UDP and TCP utilization under three different environments in Figure 10. The utilization of TCP/UDP with wireless and DSL connections are almost the same. However, for university LAN connected node, TCP utilization is much higher (10%) than other two cases. The reason causes such a result is still unclear, however, we guess that Joost provides a mechanism to avoid TCP congestion in wireless or low speed connections.

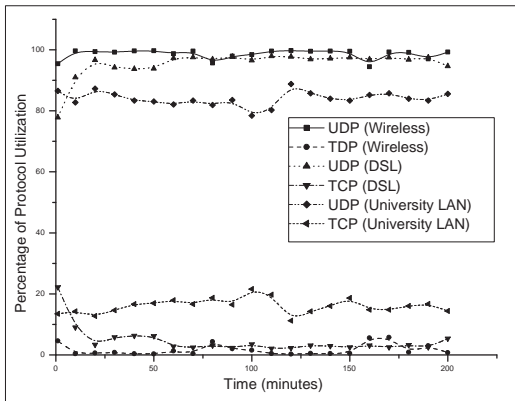


Fig. 10: UDP vs. TCP utilization.

HTTP is used for checking the software version and channel updating during bootstrapping and initialization stages. It appears when the JC browses the channel as well since the channel graphs and list are real-time downloaded from the server.

Differently, HTTPS is used for the administrative management which includes checking software version, channel list updating, obtaining tracker during reconnection and service management. For example, when the JC searches for a specific

channel during browsing, such a request will be sent to the backend server over HTTPS.

VI. MEASUREMENT METHODOLOGY

So far, we obtained the Joost architecture and described some of its functions. However, some of the mechanisms behind Joost are still unknown, such as, the locality awareness, peer selection and bandwidth capacity. In this section, we designed three typical scenarios to investigate them.

A. Experimental Conditions

We used three test machines: two Windows XP SP2 machines and one Windows Vista machine. Our test location was in Germany.

The Windows Vista node is equipped with AMD Athlon X2 64 processor, 160G hard drive, and 1.00 GB RAM. One Windows XP machine is equipped with Intel Pentium Duo-Core 1.73 processor with 1.00 GB RAM. The other Windows XP machine is Pentium 4 2.00GHz with 1.00 GB RAM. Each machine has a 10/100 Mbps Ethernet Card and additionally, the second Windows XP test node was integrated with Intel Wireless WiFi compliant wireless LAN (802.11a/g/n).

B. Considered Scenarios

The following three scenarios were considered during our experiments.

- In the first scenario (100M LAN + 100M LAN), two Windows XP (WX) nodes were connected to 100 Mbps full-duplex university LAN and located behind a network address translator (NAT) and accordingly configured with non-routable, private IP address. WX_1 started first to choose one channel. After 3 minutes, WX_2 selected the same channel.
- In the second scenario (100M LAN + 54M WLAN), the Windows Vista node (WV_1) was connected to 1Gbps university LAN and one Windows XP node (WX_1) was connected to the Gömobile Wireless LAN (54Mbps). Gömobile is a German radio network providing Internet access services. WV_1 started 20 minutes earlier than WX_1 to play the same channel.
- In Scenario 3 (DSL), one WX_2 test node was connected to a residential Arcor (a German Telecommunication company) cable-modem with 1 Mbps of downlink capacity and 1 Mbps uplink capacity.

In each of the scenario, we used different distinct Joost user accounts on separate test machines. As stated in Scenario 1, two Windows XP nodes are physically located next to each other. Thus, it would be possible that WX_2 receives a large amount of data from WX_1 since they are located in the same access network.

In the second case, WV_1 can be assumed as the high capacity node while compared with WX_1 . If there is any mechanism in Joost considering the peer's capacity, it would be noticed through the second measurement. Besides, though they are not located in the same local network, they were geographically neighbors.

In the last scenario, it would be interesting to see if cable-modem connected node (WX_2) with lower bandwidth capacity can still receive high quality video.

C. Measurement Studies

For collecting data, we used Wireshark [6] and Omnipcap [5] that allows multiple simultaneous capture sessions for different network adaptors. Since our measurements involved different network adaptors (e.g. Gigabit Ethernet, 802.11 WLAN), Omnipcap was helpful to capture packets and analyze traffic under different circumstances. Moreover, it offered a variety of selection of build-in filters to capture the packets that satisfied certain criteria. For example, it was easy to collect data with specific IP address, protocol or port number, which has also been proved in the following measurements.

Tools like MaxMind [25] and WhereIsIP [32] were used to perform reverse country, city and ISP lookups for an IP address when Omnipcap failed to return a DNS PTR record.

1) *Scenario 1 (100M LAN + 100M LAN)*: Within this measurement, we totally collected about 14.4 GB of data, over a 3-day period (7.0GB from one test node and 7.4GB from the second). By analyzing the data, we found the following facts.

Video Distribution

In the beginning, neither of them relayed any traffic for the other. More specifically, most of their data came either from the other peers, especially from European Countries (e.g. Finland, France), or from Joost owned media servers.

After a certain while, WX_2 started to receive data from WX_1 but not much (3.3%, 10.5 MB). After 3-day experiments, we analyzed the collected data from both test nodes. However, although the two test nodes were watching the same channel and also geographically and topologically locating near each other, WX_2 only received 1.3% of the data from WX_1 (96.2 MB out of approximately 7.4 GB). This may be caused by two possible reasons: (1) locality has been considered in the initial peer selection, and is adaptive to dynamic changes (e.g. new peer joining); (2) network locality and topological locality have not been taken into consideration. However, the first hypothesis does not hold since WX_1 did not send data to any other peer in the access network where WX_2 and WX_1 were located. Therefore, we conclude that Joost has not considered the network locality or topological locality during peer selection.

Channel Switching

In the end of the experiment, WX_1 switched to a new channel and WX_2 still continuously received data from it about 0.03% of the data (2.22 MB). It provides the evidence that there is a local cache for storing the old programs. Otherwise, it is impossible for WX_2 to receive data after WX_1 switched to a different program.

2) *Scenario 2 (100M LAN + 54M WLAN)*: In the second measurement, we collected 4.0 G data over a 24 hour period (2.0 G for each test node). Through analysis, we found the following facts.

For WV_1 , the quality of video playback was good. After analyzing the network traffic, we found that the most of the

data came from Joost content servers. Among the top 19 most contributed peers, there were 9 (9 out of 19) peers belonging to Joost. From these content servers, WX_1 received 21.62% (432.5 MB out of 2.0 GB). Of all other 10 most contributed peers, except for one from Japan and two from United Kingdom, the rest of the peers came from other European countries (3 from Germany). The European peers except for that from UK contributed only 57.2 MB (2.85%), whereas the peer from Japan alone sent 9.62% data (192.4 MB). The rest of the data (70%) came from other over 500 peers, however, neither of them contributed significantly.

With the wireless connection, the quality of on-demand video was good as seen in WV_1 with seldom interruptions because the wireless signal in our test lab was constantly strong. Similar to the university LAN connected node, WX_1 received a considerable amount of the data (42.23%, 0.85 GB out of 2.0 GB) from 9 Joost content servers. Among the other most contributed peers (top 10), 8 peers came from European countries and they contributed 53.9% data (1.08 GB out of 2.0 GB). Although WV_1 were located geographically close to WX_1 , it contributed 1.8% of data (360 MB).

The test result is shown in Figure 11. The tested node is put in the middle of the figure and other connecting peers are put around. Joost content servers are listed on the right side and other JCs are put on the left side. Clearly, 9 out of 19 contributors were content servers. Among the rest of the contributors, they came respectively from Denmark, Finland, Germany, Hungary, Netherland and Sweden (8 out of 10 from European countries).

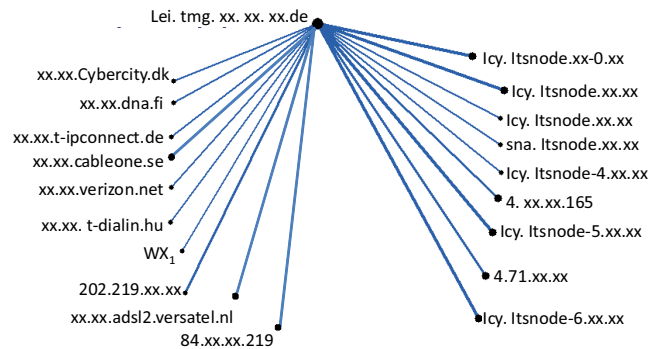


Fig. 11: Contributing peers during DSL user watching the German Channel.

There was no big difference of video quality between the WX_1 and WV_1 during this experiment, and therefore it can not be ascertained that peer's capacity has been considered in Joost. In order to verify whether Joost is sensitive to low capacity uplink and downlink, we conducted the experiments in Section VI.

3) *Scenario 3 (DSL)*: We captured five-hour period data through the residential DSL connection. Unfortunately, the quality of the video was not good (e.g. occasionally stalled) and by that time the channel was randomly selected. Therefore, we believe that Joost provides all users with the same quality of video without considering their heterogeneities. After identifying the peers providing media content to the test node, we found that most peers were cable users.

“Most Popular” Channel

After 30 minutes, we switched to a program selected from the “most popular” list. At that moment, the quality of the video was much higher than the previous channel although some of the connecting peers were still cable users. More specifically, the most contributed peers were not cable users, instead, they were high-capacity clients besides content servers.

Local Channel

In the end, we switched to a German channel and the quality was unbelievably good as there was no noticeable delay and almost no interruption. By then, the most contributed peers (top 19, ranking according to the total bytes) are almost all Joost content servers (11 out of 17, 64.7%) instead of JCs.

To our experience, two possibilities caused such a result: (1) the selected channel was rarely watched. For example, our test node was one of the few peers requesting such a channel during that time. Since there was few available JCs, the only way was to serve the client all by content servers; (2) during the peer selection, low capacity peers (i.e. WX_2) was pushed out of the existing distribution session. Later, we found the second assumption could not hold in this case since there were other JCs contributed to the test node, which were not DSL users. However, the majority of the data were still sent directly from content servers (63.38%, 290 MB).

D. Measurement Analysis

Then, we detail the three mechanisms observed during the above three experiments in the following section.

1) *Locality Considerations*: In order to find the correlation between the geographical distance and amount of data, we parsed the IP address of connected media server and other peers from which our test nodes received data. Finally, we identified 1210 distinct peers providing contents to our test nodes. These peers were located in over 54 countries. Of all the data collected from the test nodes, 45% (547) came from European countries, 24% (293) came from United States, 8.2% (99) came from Asian countries, 7.9% from South America, 3.6% from other countries. Besides, there was 45 IP addresses were not traceable and therefore we marked them as “unknown”.

Figure 12 shows that the major sources of peers are Europe and United States. Meanwhile, sources of JCs from Germany were 130 (19% of Europe). Since our host was located Germany, we conclude that the geographical distance (e.g. from specific continent) may have been considered in Joost. For example, the prefix awareness may have been considered during the peer selection. However, from Scenario 1 we ascertain that the network/topological locality has not been considered yet.

It is known that there are three main media server clusters over the world: one United States, two in Europe (one in United Kingdom) [1]. In the best case, when a client sends a video request, the request will be directed to the nearby servers. For example, a European client receives the media mostly from the European servers.

In order to further investigate the locality awareness in Joost, we measured the RTT from the receiving peers to

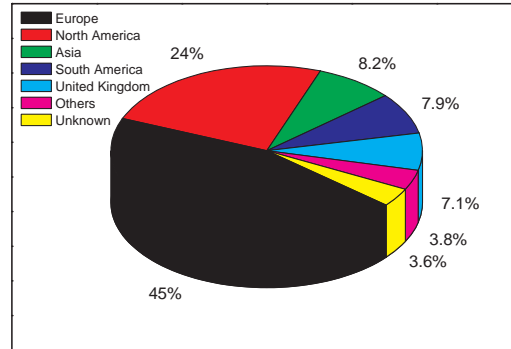


Fig. 12: Geographic Location.

the transmitting peer by OmniPing [27]. We conducted this experiment in parallel with the ping experiment by using WhereIsIP [32] to determine the number of hops between our Joost client and the transmitted peers.

TABLE II: Locality Experiments with RTT, Hops and Data

Host	Hops	RTT (ms)	Data (%)
Host 1	11	19.20	0.01
Host 2	12	113.63	0.02
Host 3	14	110.13	0.07
Host 4	13	97.397	0.48
Host 5	21	134.14	0.66
Host 6	16	128.22	0.95
Host 7	19	128.97	6.36
Host 8	16	147.14	7.22
Host 9	22	186.27	8.64
Host 10	11	416.68	8.25
Host 11	18	182.47	10.15
Host 12	21	56.19	18.07
Mean	16.17	143.36	5
Median	17.5	131.18	3.655
Standard Deviation	3.848	93.873	5.458
Correlation to Data	0.5228	0.2173	

Table 2 summarized the results of the experiments that tested peer connected with University LAN over 1 Hour 30 minutes. We firstly selected 19 peers who contributed most data to our test node. Meanwhile, there were 7 Joost content servers (7 out of 19). Then, we traced the rest 12 peers with their RTT and hop counts. As depicted in the Table, the hop counts varies ranging from 11 to 22, and the largest contributor has large hop counts. All above hosts show a weak positive correlation between RTT and the amount of transferred data. Thus, we conclude that Joost selecting peers is unlikely based on topological locality. Otherwise, the result should show strong negative (over 0.7) correlation between them.

There is a software released [22], by which user can determine which Joost-owned content distribution servers to use. How the neighboring peers can be selected is still unclear. But one thing can be assured is that topological locality is not the main metric for the peer selection algorithm in Joost.

2) *Bandwidth Capacity*: As indicated in Scenario 3, JCs with different bandwidth support may have different level service experience. As Joost dominates the network control,

it is impossible for clients to control incoming and outgoing bandwidth except for the strategic clients. Strategic clients can manually manipulate their bandwidth with help of additional traffic shaping tools. In order to identify the capacity impacts on the Joost system, we used “Traffic Shaper XP” [4] to intentionally control the bandwidth under Windows XP.

Upload Capacity Here, we changed the uplink capacity if the download capacity is full 100 Mbps. We obtained the following results.

- When the total upload bandwidth was up to 64kbit/s, the channel list could be updated but slowly and the VoD functions (e.g. fast forward) were performed bad.
- The connection with the backend server could be established when the TCP limitation was beyond 40kbit/s. That is, under this condition the request could be sent to the server and the client was ready to receive contents.
- When the UDP upload capacity increased up to 10kbit/s, the video showed and continuously played. It could be noticed that the quality of the video is fine.
- If the whole upload bandwidth was limited under 5kbit/s, there was no chance to watch program, even for updating its program list.
- If we controlled UDP upload capacity under 5kbit/s, the program list could be updated and the channel started for a while but it stopped after 5s anyway.

Above studies indicate that the uplink capacity has little impact on the video performance since with 10k bps (out of 100 Mbps) upload support JC can playback program without interceptions. That is, Joost client is assumed to contribute as much as it can without any mechanisms to encourage more contributions. Besides, it is also not possible yet because of the fixed video bit rate.

Download Capacity During the following tests, the upload bandwidth was independently unlimited.

- The quality of channel was fine without any interruption when the download bandwidth was allowed up to 1 Mbps.
- In case the download bandwidth was limited up to 512kbit/s, the program occasionally stalled.
- When the download capacity was only 128kbit/s, it was hard to continuously watch the program.
- If the download capacity was 64kbit/s, the program was no longer available.

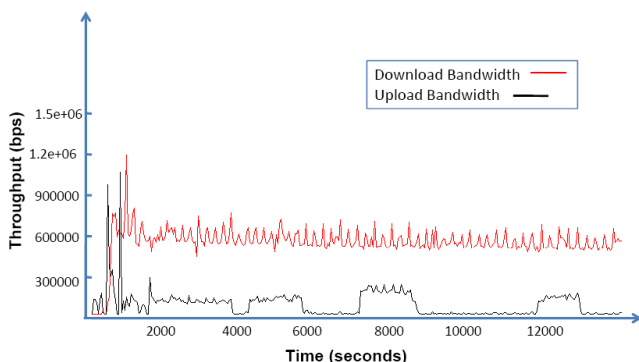


Fig. 13: Throughput Behavior of LAN Client.

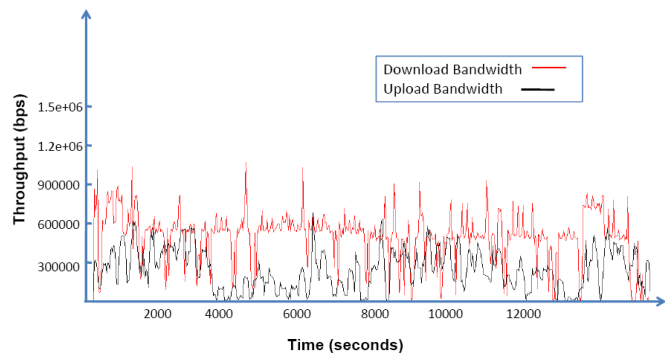


Fig. 14: Throughput Behavior of Wireless Client.

Furthermore, we measured the throughput behaviors during Scenario 2. In Figure 13, the red line represents the download capacity of WV_1 and the black line is the upload capacity. There was a large amount of throughput at the beginning of the experiment due to the initialization procedure. For both testing nodes, Joost used approximately 600 kbps down and 150 kbps up bandwidth. Regardless of the type of connections, the throughput of JC remains consistently if the client is online.

If the JC watched the program one hour, 200 – 300 MB data was received and around 100 MB was sent to other peers. Noticeably, only a few nodes (around five) are selected to be significantly served, which consume 85% upload data of the client.

3) *Peer Selection*: According to the results from three scenarios, we can deduce the peer selection mechanisms in Joost as follows.

Firstly, the popularity has been considered during the peer selection. Most likely, the JC receives a peer list with more available clients if a most popular channel is chosen.

Secondly, reconsidering the Scenario 3 we conjecture that during the peer selection, low capacity peers probably connected mostly with low capacity peers except for most popular program. It is quite similar to the swarm mechanisms used in BitTorrent [10], which allow low capacity can only receive data from comparatively low capacity node since, otherwise, high capacity node may waste resource for sending data to low capacity nodes instead of high capacity nodes. But for the most popular channels, there are enough peer resources so that it would be no problem for low capacity node to receive data from some high capacity nodes.

Thirdly, we further conclude that if the JC requests a seldom channel or it is the only one in that channel, most of the data will be sent from the Joost content server since there is not enough peers which can contribute to the client.

Fourthly, based on Scenario 1 and 2 we can conclude that the geographical locality may have been considered in Joost, however, AS-level awareness or topological locality have not been realized in the current version.

VII. RELATED WORK

Peer-to-Peer IPTV architecture requires a minimal infrastructure support and can offer the possibility of rapid deployment at low cost. In terms of simultaneous users, one

of the most successful IPTV deployments has employed P2P streaming architecture. Hei *et al.* [13] provided an overview of P2P streaming system (e.g. PPLive [3]) and characterized P2P IPTV behavior and traffic profiles at packet, connection and application levels. Among most popular IPTV services, Video-on-Demand (VoD) provides video, audio and data service triggered by users' selection. However, most of existing work about P2P VoD systems was concentrated on the protocol design and the implementation [11], [14], [12]. Different from them, we provides a real measurement analysis on Joost functions, peer selection and locality awareness. Furthermore, the Joost architecture is quite different from, even more complex than, media streaming architecture described in [13]. In Joost, each JC is more "selfish" since it only cares about the content behind its current playback position. The VoD functionalities give the users more flexibilities, and hence make the system more difficult to analyze.

Joost uses some similar P2P technologies as used in Skype which critically depends on the a peer-to-peer network formed by super nodes. Any participating node initially is a standard node, and some of them will be promoted to super nodes according to a number of factors including spare bandwidth and public reachability. Baset *et al.* [29] analyzed various aspects of the Skype protocol such as login, NAT and firewall traversal, call establishment, media transfer, codecs and conferencing under three network setups. In general, the paper provided a detailed analysis of Skype user experience and peer behaviors. Guha *et al.* [30] analyzed node dynamics and churn in Skype's peer-to-peer overlay. Further, it identified that Skype was fundamentally different from earlier P2P systems like P2P file sharing networks. There are three main differences between Skype and Joost. First, Joost architecture requires more than a login server. Second, Joost super nodes are not responsible for relaying traffic to standard nodes. Third, as observed in [29] the voice packet size varied between 40 and 120 bytes, however, the Joost video packet size was much larger (1104 bytes). Joost analysis may help to understand how P2P technologies for such VoD services should be provisioned.

Hall *et al.* [8] provided a measurement study of Joost in May, 2007. This paper explained an understanding of Joost's application behavior, network behavior, and peer behavior. However, there are several major differences between their work and our work. First, their experiments were taken based on Joost version 0.9.2 which is already out-of-date. Differently, our experimental studies were performed by Joost beta 1.0 which is more stable and integrated version. Second, through our analysis we inferred the Joost architecture and key components, however, [8] did not provide such information. Third, we designed three typical scenarios in order to further investigate the performance of locality awareness, bandwidth capacity and peer selection. Nevertheless, [8] only examined the locality awareness through three experiments. Lastly and more importantly, we analyzed the Joost VoD functionalities which are the main difference from other media streaming systems. Therefore, we can argue that our paper provides the first comprehensive analysis of Joost P2P VoD service.

VIII. CONCLUSIONS AND FUTURE WORK

Joost is one of the first commercial Peer-to-Peer VoD systems which can provide high quality on-demand TV based on P2P technologies. We have attempted to discover various aspects of the Joost functions and behaviors by analyzing the network traffic and by being acquainted with some of the open software used in Joost. Without a surprise, Joost and Skype have some P2P mechanisms and techniques in common. Our major contributions include: (1) we inferred the Joost architecture and some key components based on careful studies of Joost network traffic; (2) we further took a close investigation on its media streaming behaviors and peer management behaviors; (3) with three envisioned typical scenarios we have further studied the performance of locality awareness, bandwidth capacity and peer selection. To our best knowledge, this paper is the first comprehensive analysis on Peer-to-Peer VoD services.

Overall, our study demonstrates that with some dedicated infrastructure the current Internet infrastructure is capable of providing performance requirements of high quality VoD. Based on extensive measurements, we infer that Joost is a server-assisted peer-to-peer VoD system. Joost mainly relies on plenty of dedicated infrastructure nodes (e.g. content servers) to distribute video. The P2P technologies are used to help distributing video and to extend the system's scalability.

Although large-scale P2P VoD systems are feasible in today's Internet, the performance remains to be improved in the following branches:

- Such a architecture heavily relying on a set of centralized content servers may still raise a scalability issue in the near future. Currently, the scalability is not a major issue due to the limited number of users.
- Joost has not efficiently used the peers' resources, especially when the high capacity peers are available. In other words, Joost could be slightly more aggressive with uplink resource of high capacity peers. For example, it takes a long time to browse the channel list since it is dynamically downloaded from the server. If there is a crowded browsing, the server is highly overloaded. These high capacity peers could be used for providing such a service.
- It was noticed that during our experiments there were over five times of "This program is unavailable right now". When it happened with one program, it happened with all programs. After waiting for a few minutes, even up to 30mins, the programs started. Therefore, we believe that the current Joost P2P technology isn't always reliable.
- Joost currently provides each client with the same quality of video, which can be deduced from the results of above scenarios. This may result in an inefficient resource utilization if some clients are unable to support the desired video quality. Hence, layered video or adaptive mechanisms, together with certain incentive mechanisms, may be introduced into Joost.

This paper provides a first trial on investigating the Joost peer behaviors and media distribution mechanisms. Current Joost P2P code maybe neither AS-level aware nor end-to-

end latency aware for the peer selection. However, the exact peer lookup and selection techniques that Joost used for peer management is still not clear. Our guess is that it uses a combination of swarm techniques in BitTorrent and prefix awareness. Therefore, we intend to investigate them in the next stage, for example, the peer selection, local cache and on-demand video streaming. Moreover, we noticed the significant difference of TCP utilization in different network connections (Section V) and therefore, we decide to trace the TCP network traffic of Joost clients especially in the wireless environment.

REFERENCES

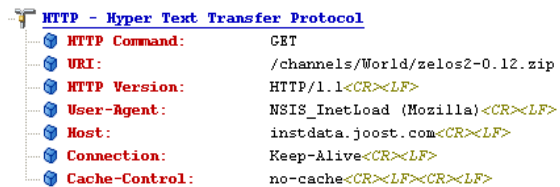
- [1] "Joost," <http://www.joost.net/>.
- [2] "Joost Network Architecture," <http://www.scaryideas.com/video/2362/>.
- [3] "PPLive," <http://www.pplive.com>.
- [4] "Traffic Shaper XP," <http://bandwidthcontroller.com/trafficShaperXp.html>.
- [5] "WildPackets," <http://www.wildpackets.com/products/omnippeek/overview>.
- [6] "Wireshark," <http://www.wireshark.org>.
- [7] C. L. Abad, W. Yurcik, and R. H. Campbell, "A survey and comparison of end-system overlay multicast solutions suitable for network-centric warfare," In Proceedings of the SPIE Battlespace Digitization and Network-Centric Systems IV, Vol. 5441, pp. 215–226, 2004.
- [8] Y. J. Hall, P. Piemonte, and M. Weyant, "Joost: A Measurement Study" Carnegie Mellon University, May 14, 2007.
- [9] B. Bhargava, C. Shi and S.-Y. Wang, "MPEG Video Encryption Algorithms," *Multimedia Tools and Applications*, Vol. 24, No. 3, pp. 57–79, April 2004.
- [10] B. Cohen, "Incentives build robustness in bittorrent," in Proc. of 1st Workshop on the Economics of Peer-2-Peer Systems, Berkley, CA, 2003.
- [11] T. Do, K. Hua, and M. Tantaoui, "P2vod: providing fault tolerant video-on-demand streaming in peer-to-peer environment," In Proc. of IEEE ICC 2004, Paris, France, June 2004.
- [12] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2cast: peer-to-peer patching for video on demand service," *Multimedia Tools Appl.*, vol. 33, no. 2, pp. 109–129, 2007.
- [13] X. Hei, C. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," to appear in *IEEE Transactions on Multimedia*, Nov. 2007
- [14] C. Huang, J. Li, and K. Ross, "Peer-assisted vod: Making internet video distribution cheap," in Proc. IPTPS 2007, Feb. 2007.
- [15] I. Norros and B.J. Prabh and H. Reittu, "Flash crowd in a file sharing system based on random encounters." Proc. of ICST/ACM workshop on Interdisciplinary systems approach in performance evaluation and design of computer & communications systems, 2006.
- [16] Internet Assigned Numbers Authority (IANA), <http://www.iana.org>.
- [17] ITU Telecommunication Standardization Sector (ITU-T), <http://www.itu.int/ITU-T/>.
- [18] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Draft (draft-ietf-mmusic-ice-17), IETF, 2007.
- [19] J. Rosenberg and J. Weinberger and C. Huitema and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, IETF, 2003.
- [20] J.M. Almeida and D.L. Eager and M. Frris and M.K. Vernon, "Provisioning Content Distribution Networks for Streaming Media," In Proc. of INFOCOM 2002.
- [21] Joost Open Source, <http://opensource.joost.net/>.
- [22] Joost Support Forum, <http://www.joost.com/support/faq/Technology.html>.
- [23] Kazaa, <http://www.kazaa.com/>.
- [24] Level 3 Communications, <http://www.level3.com/>.
- [25] MaxMind, http://www.maxmind.com/app/locate_ip.
- [26] O. Babaoglu and H. Meling and A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems," In Proc. of 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02), 2002.
- [27] OmniPing Professional, <http://www.manasoft.com/manasoft/>.
- [28] R. Hipp, "SQLite," <http://www.sqlite.org/>.
- [29] S. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in Proc. of INFOCOM'06, Barcelona, Spain, 2006.
- [30] S. Guha, N. Daswani, and R. Jain, "An Experimental Study of the Skype Peer-to-Peer VoIP System," in Proc. of IPTPS 2006.
- [31] H. Schulzrinne and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control," RFC 3551, IETF, 2003.
- [32] WhereIsIp, <http://www.jufsoft.com/whereisip/>.

APPENDIX

A. HTTP messages involved in Joost installation

This section shows the message dump of HTTP 1.1 GET request that a JC sent to backend.joost.net (89.251.4.175) and the responses it received.

In case it was the first time after installation, the client sent a HTTP 1.1 GET request containing the URI of the resources. The requested file is `zelos2-0.12.zip` which can be unzipped into `zelos2-0.12.sqlite` as described in Section IV.

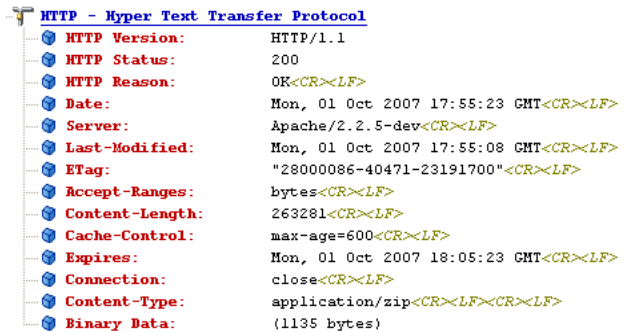


```

HTTP - Hyper Text Transfer Protocol
  HTTP Command:      GET
  URI:               /channels/World/zelos2-0.12.zip
  HTTP Version:     HTTP/1.1
  User-Agent:       NSIS_InetLoad (Mozilla)
  Host:             instdata.joost.com
  Connection:      Keep-Alive
  Cache-Control:   no-cache
  
```

Fig. 15: Joost installation: HTTP GET

Then, there was a 200 OK response received by the client for the above GET request.



```

HTTP - Hyper Text Transfer Protocol
  HTTP Version:     HTTP/1.1
  HTTP Status:      200
  HTTP Reason:     OK
  Date:            Mon, 01 Oct 2007 17:55:23 GMT
  Server:         Apache/2.2.5-dev
  Last-Modified:  Mon, 01 Oct 2007 17:55:08 GMT
  ETag:           "28000086-40471-23191700"
  Accept-Ranges:  bytes
  Content-Length: 263281
  Cache-Control:  max-age=600
  Expires:       Mon, 01 Oct 2007 18:05:23 GMT
  Connection:    close
  Content-Type:  application/zip
  Binary Data:   (1135 bytes)
  
```

Fig. 16: Joost installation: HTTP OK

B. HTTP messages involved in Joost initialization

During the bootstrapping procedure, JC sent a HTTP 1.1 GET request to `instdata.joost.com` which is actually the version server (89.251.2.87). The current software version is 0.13.0 (Beta 1.0). The fragmentation of the HTTP GET request is shown as follow.

```

HTTP Command: GET
URI:         /?version=0.13.0
HTTP Version: HTTP/1.1
Host:       instdata.joost.com
User-Agent: Mozilla/5.0 Windows; U; Windows NT 5.1; ...
  
```

Fig. 17: Joost initialization: HTTP GET

Then, a 200 OK response was sent from the version server to the client.

```

HTTP Version: HTTP/1.1
HTTP Status: 200
HTTP Reason: OK
Server: Apache/2.2.5-29
Cache-Control: max-age=600

```

Fig. 18: Joost initialization: HTTP OK

C. HTTP messages involved in Joost reconnection

This section shows the message dump of HTTP 1.1 GET request that a JC sent to channel graphs server and the responses it received.

The HTTP GET containing the URI of requesting Compressed Scalable Vector Graphics File (.svgz) is shown below. At that moment, the channel list management was redirected to lux-backend-13-bond0.joost.net (89.251.4.153).

```

HTTP - Hyper Text Transfer Protocol
  HTTP Command: GET [54-56]
  URI: /pJKty-a9I_hAwG4ySzmdLA.svgz [57-85]
  HTTP Version: HTTP/1.1<CR><LF> [86-96]
  Host: j00.st<CR><LF> [97-110]
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9a5Spre)
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Language: en-us,en;q=0.5<CR><LF> [291-323]
  Accept-Encoding: gzip,deflate<CR><LF> [324-354]
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7<CR><LF> [355-402]
  Keep-Alive: 300<CR><LF> [403-419]
  Connection: keep-alive<CR><LF><CR><LF> [420-445]

```

Fig. 19: Joost reconnection: HTTP GET

Then, there was a OK response received by the client for the above GET request.

Joost system is built on the top of several open softwares [21]. In this section, we briefly summarize some of the most related mechanisms for the better understanding of the system.

D. Anthill

Joost uses Antihill Model [26], a agent-based peer-to-peer system, to support the P2P media distribution services. An Anthill system is composed of a network of interconnected *nests* which are middleware layers capable of performing computations and hosting resources. Any machine connected to the Internet and running Anthill can act as a nest. If a nest receives a request from the local application, one or more *ants* - autonomous agent - will be generated to satisfy the request.

Logically, a nest contains three modules: ant scheduler, communication layer and resource managers. The *communication layer* is responsible for discovery of new nests, network topology management and for ant movement between nests. Each service installed by a nest is associated with a set of *resource manager* modules. For instance, the Joost file-sharing service based on a distributed index for file retrieval. A file manager is used to maintain shared files; URL manager is used to maintain the distributed index; and a routing storage is used by ants to make routing decisions.

Anthill provides a configuration mechanism that the structure of the network, the ant algorithm, characteristics of the workload are all defined in its XML files (share.xml). More specifically, we found that the the IP address, negotiated port

number (ListeningPort), inbound and outbound bandwidth, and NodeID are defined in the share.xml file.

E. STUN

Joost middlebox traversal is currently performed by STUN protocol [19] that allows a client to discover whether it is behind a NAT or firewall and the type of the NAT or firewall. It is achieved with the help of a special server in the public address space, called STUN server. For example, a JC sends an exploratory message to the STUN server and the server checks the message and informs the client about the public IP address and ports used by the NAT. By recording the public IP address and port in the share.xml file, the JC can use them to send and receive packets without intervening the STUN server.

However, STUN only works for UDP and does not work with the “symmetric NAT” which is the most common NAT type in corporate networks. Therefore, Joost is currently investigating Interactive Connectivity Establishment (ICE) [18] that allows the client to learn the topology it meets and different types of firewalls that may exist between the client and the network [2]. By doing this, the client can easily learn how to communicate with others since existing firewalls could be successfully traversed.

The snapshot of the Joost channel database in Figure 20.

URN	ID	Title	Description	Logo
urn:tv:banquet.thev	104000g	Banquet		
urn:tv:alliance_atlan	4500055	Alliance Atlantis Sci		
urn:tv:thomas_lucas	246000i	SpaceRip: Space, S		
tag.clouseau.thever	client-popular	Popular		
urn:tv:zelos:hodgep	hodge-podge	Joost Links	Joost Links are short	
urn:tv:ministry_of_so	065001u	Ministry of Sound TV	What's current in cl	https://thumbs.ops.thevenic
urn:tv:reuters_us.the	069000h	Reuters	Reuters brings you th	https://thumbs.ops.thevenic
urn:tv:banquet.thev	104000g	Banquet	Banquet, The Cham	https://thumbs.ops.thevenic
urn:tv:northone.thev	01900b6	Fifth Gear Shortcuts	Popular award-winni	https://thumbs.ops.thevenic
urn:tv:tvip.thevenic	024000h	Joost Suggests	Welcome to Joost! H	https://thumbs.ops.thevenic
urn:tv:viacom_mtv_j	0860002	MTV	MTV needs no intro	https://thumbs.ops.thevenic
urn:tv:universal_bug	136005g	Bugeye Music	Live concert footage	https://thumbs.ops.thevenic
urn:tv:alliance_atlan	0450055	Alliance Atlantis Sci	Check out the finest	https://thumbs.ops.thevenic
urn:tv:viacom_param	0910077	Paramount Pictures	Disfrutan de esta co	https://thumbs.ops.thevenic
urn:tv:viacom_param	0910079	Paramount Pictures	Geniessen Sie eine /	https://thumbs.ops.thevenic
urn:tv:viacom_param	0910078	Paramount Pictures	Retrouvez une colle	https://thumbs.ops.thevenic
urn:tv:off.thevenicp	020006n	Off the Fence Docs	Docs on demand - s	https://thumbs.ops.thevenic
urn:tv:mondo_media	203001g	Mondo Mini Shows	Mondo Mini Shows :	https://thumbs.ops.thevenic
urn:tv:gong.thevenic	01300a7	GONG	GONG is the first an	https://thumbs.ops.thevenic
urn:tv:aardman.thev	0440001	Aardman Animations	Multi award-winning	https://thumbs.ops.thevenic
urn:tv:tv5.thevenic	1660045	TV5MONDE PLUS	TV5MONDE is well	http://j00.st/MclwovJL7Gc
urn:tv:the_onion.the	163000f	Orion News Networ	The Orion News Ne	https://thumbs.ops.thevenic
urn:tv:indivisual.thev	0590003	The Soccer Channe	Relive the greatest g	https://thumbs.ops.thevenic
urn:tv:warnermusicg	08200ma	Best of Today	The Best of Today E	https://thumbs.ops.thevenic
urn:tv:warnermusicg	08200m9	Stadium Rockers	Stadium Rockers is l	https://thumbs.ops.thevenic

Fig. 20: Initial Channel List.