Evaluating a publish/subscribe proxy for HTTP

ZHANG YUANHUI

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Evaluating a publish/subscribe proxy for HTTP

Zhang Yuanhui

yuanhui@kth.se

2013.04.20

Examiner & Supervisor: Gerald Q. Maguire Jr.

KTH Royal Institute of Technology
School of Information and Communication Technology
Stockholm, Sweden

Abstract

With the increasingly high speed of the Internet and its wide spread usage, the current Internet architecture exhibits some problems. The publish/subscribe paradigm has been developed to support one of the most common patterns of communication. It makes “information” the center of communication and removes the “location-identity split” (i.e., that objects are at specific locations to which you must communicate with to access the object). In this thesis project a publish/subscribe network is built and then used in the design, implementation, and evaluation of a publish/subscribe proxy for today’s HTTP based communication. By using this proxy users are able to use their existing web browser to send both HTTP requests and Publish/Subscribe Internet Routing Paradigm (PSIRP) requests. A publish/subscribe overlay is responsible for maintaining PSIRP contents. The proxy enables web browser clients to benefit from the publish/subscribe network, without requiring them to change their behavior or even be aware of the fact that the content that they want to access is being provided via the publish/subscribe overlay. The use of the overlay enables a user’s request to be satisfied by any copy of the content, potentially decreasing latency, reducing backbone network traffic, and reducing the load on the original content server. One of the aims of this thesis is to make more PSIRP content available, this is done by introducing a proxy who handles both HTTP and PSIRP requests, but having received content as a result of an HTTP response it publishes this data as PSIRP accessible content. The purpose is to foster the introduction and spread of content based access.

Keywords: *Publish/Subscribe, HTTP proxy, Blackadder, PURSUIT, Pub/sub proxy, network metrics*

Sammanfattning

Med allt högre Internetshastighet och dess utbredda användning, uppvisar den aktuella Internet-arkitekturen vissa problem. Publicera / prenumerera paradigm har utvecklats för att stödja en av de vanligaste mönstren för kommunikation. Det gör att "information" blir centrum av kommunikation och tar bort "plats-identitet split" (dvs att objekten är på specifika platser som du måste kommunicera med för att komma åt objektet). i detta examensarbete byggs ett publicera / prenumerera nätverk och sedan används i utformningen, genomförandet, och utvärdering av en publicera / prenumerera proxy för dagens HTTP-baserad kommunikation. Genom att använda denna proxy kan användare kan använda sin befintliga webbläsare för att skicka både HTTP-förfrågningar och publicera / Prenumerera Internet Routing Paradigm (PSIRP) begäran. En publicera / prenumerera överlagring är ansvarig för att upprätthålla innehåll av PSIRP. Fullmakten gör det möjligt för klienter av webbläsare att dra nytta av publicera / prenumerera nätverket, utan att kräva dem att ändra sitt beteende eller ens vara medvetna om det faktum att det innehållet som de vill komma åt tillhandahålls via publicera / prenumerera överlägg. Användningen av överlägget kan en användare begäran som skall uppfyllas av en kopia av innehållet, eventuellt minskande latens, vilket minskar trafiken stamnät, och minska belastningen på det ursprungliga innehållet servern. Ett av syftena med denna uppsats är att göra mer PSIRP innehåll tillgängligt och detta görs genom att införa en proxy som hanterar både HTTP och PSIRP förfrågningar, men har fått innehåll som en följd av en HTTP-svar offentliggörs denna data som PSIRP tillgängligt innehåll. Syftet är att främja införandet och innehållbaserade åtkomsten.

Nyckelord: *Publicera / Prenumerera, HTTP-proxy, Blackadder, PURSUIT, Pub / sub proxy, nätverk statistik*

Acknowledgements

I would like to express my greatest gratitude to the people who have helped and supported me throughout my Master thesis. Firstly, I am grateful to my supervisor Professor Gerald Q. Maguire Jr. for his continuous support and valuable comments for the project. And I wish to thank my parents for their kindness and encouragement throughout my study. Last but not least, I would like to express my very appreciation to my friend Zhang Wen for his valuable and constructive suggestions during the planning and development of this thesis work.

Table of Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AID	Application level identifier
API	Application programming interface
CCN	Content-Centric Networking
DDoS	Distributed denial of service
DHR	Document Hit Ratio
DHT	Distributed hash table
DiffServ	Differentiated services
DNS	Domain Name System
DONA	Data-Oriented Network Architecture
DoS	Denial of service
FID	Forwarding Identifier
FNV	Fowler/Noll/Vo
FTP	File Transfer Protocol
HI	Host Identifier
HIP	Host Identity Protocol
HTTP	Hypertext Transfer protocol
ID	identifier
IPsec	IP security
NNTP	Network News Transfer Protocol
PSIRP	Publish/Subscribe Internet Routing Paradigm
Pub/Sub	Publish/Subscribe
RH	Resolution Handler
RFC	Request for Comments
RID	Rendezvous identifier
ROFL	Routing on Flat Labels (ROFL)
RP	Rendezvous Point
RRT	Request Response Time
RSS	Really Simple Syndication
SID	Scope identifier
SMTP	Simple Mail Transfer Protocol
URL	Uniform Resource Locator
VoIP	Voice over IP
WWW	World wide web

1 Introduction

This chapter begins with a general introduction to the current Internet architecture, as well as the problems it faces. The second section introduces the thesis project's goal, which is to build a publish/subscribe network and to design, implement, and evaluate a publish/subscribe proxy for what is today HTTP based communication. The chapter ends with a summary of the structure of this thesis.

1.1 Current Internet and Issues

In the current information technology society, the Internet is widely used in various fields, and it is rapidly developing. During the last 20 years, the basic Internet architecture has not undergone any major changes. Today it is very hard to completely change the core protocol, as there are so many users and businesses that depend upon the network's operation. Additionally, there is no longer any central control, so a global change in protocol is difficult. Moreover, the increasing use of information and communication technology requires reliable underlying networks – at least the aggregate performance should be reliable. The Internet today is based on a host-to-host communication paradigm and is packet-oriented. However, with the dramatically increasing numbers of users and ever increasing amounts of traffic, the network environment and protocols are becoming more and more complex. This architecture faces some serious problems, including scalability, exhaustion of the IPv4 address space, a massive increase in traffic due to the spread of on-demand video, etc. Also, many security issues have to be considered. Some of the threats to security include distributed denial of service (DDoS) attacks, spam, viruses, Trojan horses, and phishing. Moreover, these security threats are a growing problem. The result is that today the Internet is a challenging and insecure environment. Therefore, some evolutionary approaches have been proposed and are in various stages of development to address these problems.

IPv6 addresses are 128-bit identifiers for interfaces. IPv6 supports unicast, anycast, and multicast. It also has improved support for extensions and options, with the IPv6 options placed in separate headers that are located between the IPv6 header and the transport layer header. Another improvement is that a new function has been developed to enable the labeling of packets belonging to particular traffic flows for which the sender requests special handing, such as a specific quality of service. IPv6 requires that network layer security (IPsec) be part of every implementation. IPsec can be used to carry data traffic. Today developers only use IPsec's encapsulating Security Payload (ESP) to carry encrypted data packets.

The Host Identity Protocol (HIP) [1] defines a new global Internet name space, which fills an important gap between the IP and DNS namespaces. The Host Identity namespace consists of Host Identifiers (HIs). HIs decouple the name and locator roles. Each host will have at least one Host Identity, but a host will typically have more than one HI. Each HI uniquely identifies a single host. In this new approach the transport layer operates on HIs instead of IP addresses, while the network layer uses IP addresses as pure locators.

Differentiated services (DiffServ)[2] are intended to provide a framework and building blocks to enable deployment of scalable service discrimination in the Internet. The differentiated services approach aims to speed deployment by separating the architecture into two major components. One component is the fairly well-understood behavior of the current forwarding path and the other is the more complex and still emerging policy and allocation component that configures parameters used in the forwarding path. DiffServ is not hard to deploy and offer good performance for Voice over IP (VoIP) and real-time video. However, DiffServ is not widely used *between* operators.

A distributed hash table (DHT) is a decentralized distributed system that provides a lookup service similar to a hash table. In this approach key-value pairs are stored in a DHT. Any participating node can efficiently retrieve the value associated with a given key. A DHT can be used to implement distributed databases or content delivery systems.

The Internet of today is a platform for information exchange and spreading digital content. Because of the difference between the original Internet's design and its actual usage, a new Internet architecture based on a content-centric paradigm has been proposed. Such a content-centric communication paradigm could address problems and challenges related to scalability, quality of service, security, and mobility – as this approach decouples the object from specific locations, thus any copy of the object offers (read) access. It should be noted that the main focus of content centric networking is on read access to objects that once created do not change (for example, e-mails, audio clips, video clips, photographs, etc.).

The Content-Centric Networking (CCN) [28] architecture intends to build a communication system that encourages users to focus on their desired content. The user does not care about the physical location of their desired content. An implementation of CCN uses a data storage cache to avoid duplicate up-stream traffic. Once information is stored in the cache of one of the nodes, the response time will decrease when a second request is made for this same content. This approach is much more efficient than having a centralized server provide the content. In order to locate specific content, each CCN packet must have a unique human-readable name that can sufficiently define the desired content. The name should contain version information (to correctly identify the data), a cryptographic hash (to authenticate the content), and an indication of the decoding format (so that the user can indicate what format(s) they can decode). There are two types of CCN packets: one of them is an interest packet and the other one is a data packet. Data is transmitted to the user *only* if the subscriber sends an interest request (thus preventing spam and avoiding DoS attacks). The two types of packets have the content name inside. When an instance of the data satisfies an interest with a longest prefix name match, then the data packet will be delivered to the subscriber.

The publish/subscribe Internet Routing Paradigm (PSIRP) aims to implement publish/subscribe networking without relying on existing networking protocols. This paradigm changes the control from a data sender to the potentially many data consumers. Information is propagated only to a user who requests it. Information is the core element of the PSIRP architecture; everything is information and information is everything [3]. Information is organized in a hierarchical way, so small 'meaningless' pieces of data, which can be arbitrary chunks of data, are combined into large complex information items such as files, documents, pictures, and videos. An information item may be used as a reference to other items, providing information such as data size, information owner, permissions, and composition elements[4]. An information scoping mechanism is introduced to limit reachability; this allows the data source to limit the reachability of information to only those parties having access to the particular mechanism that implements the scoping. Scopes can be considered as the equivalent of IP topologies, because scoping mechanisms allow the realization of information networks, while the IP structure builds an inter-networking topology based on IP addresses. Every piece of data belongs to at least one scope. For example, a scope "TV show" might not only belong to the scope "Science", but also to the “College Education” scope. When new content becomes available, the publisher signs a scope identifier. For identification, a unique label identifies every piece of information. This label is used in subscriber interests to match against published information. The function of matching a subscriber’s interests with published information is known as the **rendezvous function** and for this reason this label is referred to as the **rendezvous identifier** (RID). A subclass of rendezvous identifiers is the **scope identifier** (SID). A SID denotes the specific scope within which the information is reachable. RIDs and SIDs are independent of the endpoints that produce and consume the associated information items. Flat endpoint independent labels seem to be a natural choice for information oriented architectures, as they clearly separate location from identity — allowing for mobility, multicasting, and multi-homing in the architecture[5].

For anyone who uses the Internet, the World Wide Web (WWW) must be a familiar thing. A WWW connection request utilizes the Hypertext Transfer protocol (HTTP)[6]. .HTTP is an application protocol for distributed, collaborative, hypermedia information systems, thus HTTP is the foundation of the World Wide Web (WWW). In HTTP there are two parties: the client directly connects to the server and the server responds to the client’s requests. Additionally, we can insert an HTTP proxy into this communication path. The proxy acts on behalf of a client by acting as a server, while it in turn acts as a client to a HTTP server. Proxies were designed to provide a gateway to enable access to a website for people in closed subnets, who otherwise would only be able to access the Internet via a firewall machine. A client first sends a URL to the proxy; the proxy then connects to the destination server. After receiving a response from the server the proxy forwards the results back to the client. The proxy server can also download and store the response in a local cache. If there is a subsequent request for this same URL and this content has already been stored in the cache of the proxy, then the proxy can immediately return the data rather than sending the request to the target server. This caching of retrieved documents reduces the access latency. However, if the object might change, then there is a question of if the cached version is up-to-date or not – this may require that the proxy ask the server if the content has been modified since the time when it was cached, if not then the cached content is valid, otherwise the proxy will request a copy of the current content. This proxy architecture with caching is shown in Figure 1-1.

The HTTP proxy can also implement a number of functions, including filtering individual HTTP requests and responses, characterizing sets of massages, keeping machines behind it anonymous for security reasons, speeding up access to the resources (using caching), accessing sites prohibited or filtered by some ISP or institution, applying access policies to network services or content, etc.

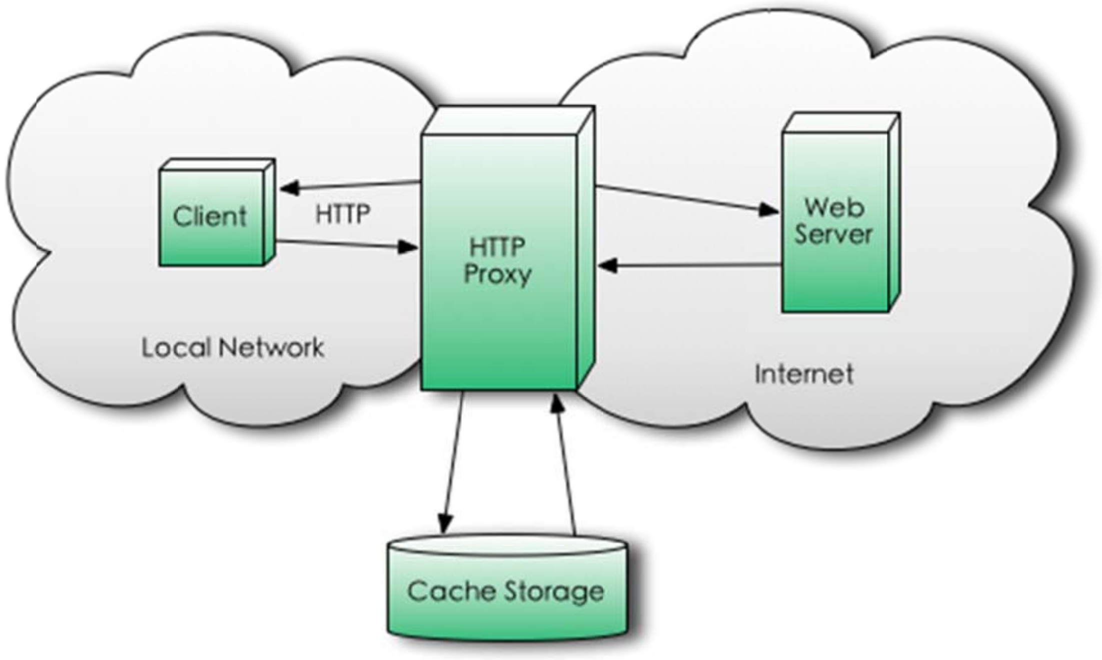


Figure 1-1: A HTTP proxy connects to the local network and remote server

1.2 Project’s Goal

The goal of this project is to design, implement, and evaluate a publish/subscribe proxy for HTTP that has two functions combined together. One function is to act as a normal proxy (accepting HTTP requests and proxying them to the target web server). The second function is **that the proxy allows a client to query for a publication through a publish/subscribe system with PSIPR URLs** and then the proxy returns the content to the client. As shown in Figure 1-2, the client has two options to obtain services or resources: sending an http://… or a psirp:// request to the proxy. When the proxy receives information from a remote server or publisher, it forwards packets directly to a client **and** saves the content in its cache. One of the most important abilities of this proxy is to **generate and cache** PSIRP content, thus the proxy could publish any of the content it receives as PSIRP content to be accessed by other clients. To be specific, the proxy transforms the content addressed by HTTP URLs into content accessible via the psirp URL scheme. The format of a psirp: URL is:

psirp://[scope identifier]: [rendezvous identifier]

These “psirp:” URLs can be used in the exactly same way as any other URL. For example, they can be typed into a browser’s address bar.

In this project, we decided to use an existing PURSUIT prototype, which has been implemented for the Linux operating system. This prototype includes the basic pub/sub functionalities and zFilter based forwarding. Both local and remote publications use the same paradigm. A PSIRP plug-in for the Firefox web browser has been developed to use the pub/sub stack.

We intend to design a cache in the pub/sub proxy to store useful information. How to choose and identify useful chunks of information needs to be evaluated during the design and evaluation process. For example, when a client asks for a real-time video chat, it will likely be useless to store this video in the cache. For this reason, the pub/sub HTTP proxy must have the ability to identify the content and select what content is to be cached.

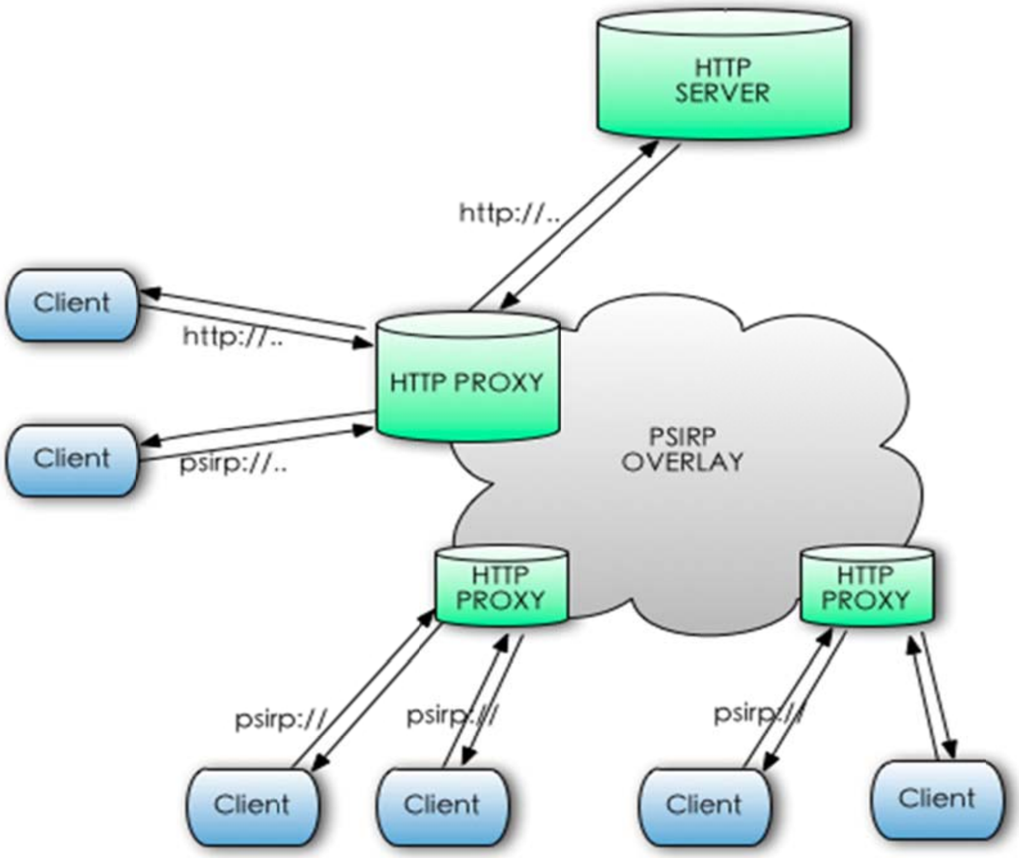


Figure 1-2: The topology of Pub/Sub proxy for HTTP

1.3 Thesis structure

Following this first chapter, Chapter 2 introduces the reader to related work and introduces the Publish/Subscribe Internet Routing Paradigm (PSIRP). Details of PSIRP are given in Chapter 3. Chapter 4 describes how a HTTP proxy can enable HTTP web browser clients to utilize PSIRP. The design of a prototype proxy is given in Chapter 5. Chapter 6 describes the design and implementation of this prototype proxy. Chapter 7 presents the results of evaluating this prototype. Chapter 8 summarizes the conclusions based upon the results discussed in Chapter 7, and then suggests some future work. It also concludes with some reflections on the social, economic, and ethical aspects of this thesis project.

2 Background

In the global Internet, the current IP model establishes a routing fabric centered on a topological notion of the network. Packets are delivered end-to-end between two explicitly addressed endpoints. To design and build a more adaptive network architecture experts have proposed alternative architectures. A variety of changes have been suggested based upon technical, social, and business arguments. In this section, we will introduce some related prior work regarding the publish/subscribe (pub/sub) architecture that has been suggested as means for facilitating the distribution of information. This pub/sub architecture and the need for current users of web browsers to access information, while taking advantage of the pub/sub architecture, is the motivation for this thesis project.

2.1 Related work

This section introduces two of the most relevant examples of prior work: the Data-Oriented Network Architecture (DONA) and Routing on Flat Labels (ROFL). Each of these will be described in more detail in the following subsections.

2.1.1 DONA

The Data-Oriented Network Architecture (DONA)[7] replaces the hierarchical Domain Name System (DNS) name space with a system of flat, self-certifying names, and replaces DNS name resolution with a name-based anycast primitive that operates *above* the IP layer[15]. DONA improves data retrieval and service access by providing stronger and more architecturally coherent support for persistence, availability, and authentication. DONA can also be extended to provide support for caching and Really Simple Syndication (RSS) like updates. DONA has a strict separation between naming (persistence and authentication) and name resolution (availability)[7].

DONA names are organized around principals. Each principal is associated with a public-private key pair, and each datum, service, or any other named entity (host, domain, etc.) is associated with a principal. The form of the name is P: L where P is the cryptographic hash of the principal’s public key and L is a label chosen by the principal (who ensures that the name is unique). DONA uses the route-by-name paradigm for name resolution. Rather than using DNS servers, DONA relies on a new class of network entities called resolution handlers (RHs)[7]. Name resolution is accomplished by using two basic primitives: FIND(P: L) and REGISTER(P: L). A client emits a FIND(P: L) packet to locate the object named P: L, and RHs route this request towards a nearby copy. The REGISTER message is used to inform the RHs of where copies of information are so that they can route FIND requests effectively.

2.1.2 ROFL

Routing on Flat Labels (ROFL) is routing protocol based on the idea of a distributed hash table (DHT). ROFL routes based upon host identities rather than physical addresses. In order to send a packet, the client first adds an initial source route to the packet. This initial source route points the packet to the router with the “Closest” ID, where closest is assessed in terms of the destination host’s ID. This route leads the packet to a router whose ID does **not** exceed the destination host’s ID. During transmission, if another router has a more specific and complete route, i.e., one which reaches a router closer to the destination, then this router will replace the original route with the better route. ROFL utilizes a flat namespace, where the names have no semantic content. This approach inherits the advantages of the location-identity split, such as mobility, multihoming, and stable identities, but also has several practical advantage of its own. One of these advantages is that developers do not need to build a separate name resolution infrastructure. The second advantage is fate-sharing, which means that packet delivery does not depend on anything off the data path – since there is no longer any need to contact a resolution infrastructure before sending a packet. Compared to IP addresses, this model has simpler allocation system as it only requires uniqueness of identities, while IP addresses must be both unique and adhere to the network topology. Last but not least, ROFL has more appropriate access controls that are based on identifiers [16]. Note that a disadvantage is that one can no longer know where the problem is when a packet is not routed to the desired host, since the responsibility for forwarding is distributed over all the routers that might be on a path from the source to the destination. Additionally, since the uniqueness of name is still required, but no longer administratively defined, the name space must be sufficiently large that there is a low probability of their being a name collision.

It might be noted that there are other proposals for flat name spaces, such as HIP, AIP, and MobilityFirst. There are also other mapping proposals, such as DHT-MAP, SLIMS, and DMap [8].

2.2 PSIRP

Two European telecommunication vendors and several academic institutions have investigated publish/subscribe (pub/sub) approach. The resulting Publish/Subscribe Internet Routing Paradigm (PSIRP) [9] approach aims to build an Internet architecture around the information and to give control to the *receiver*. In this architecture the receiver is in control since this node can choose which information it wants to receive. In this approach no node receives any content to which it has not explicitly expressed an interest in by way of a subscription. This information centric pub/sub communication architecture decouples the sender (i.e., publisher) and the receiver (i.e., a subscriber) in time, space, and context by putting an explicit publication operation in middle. This basic architecture is shown in Figure 2-1. In the pub/sub model, the act of publication produces a persistent, immutable association between an identifier and a data value (the publication) created by the publisher. By knowing the identifier, a subscriber can search for desired information through the PSIRP network. If subscribers keep asking for some specific topic or information, we could number the events and encode this information in the identifiers, which will make it easier to fetch the content. The self-certifying identifiers can be utilized to bind the content securely to the identifiers. We can use an intermediate publisher to implement channels that have multiple publishers. Further details of PSIRP will be given in the next chapter.

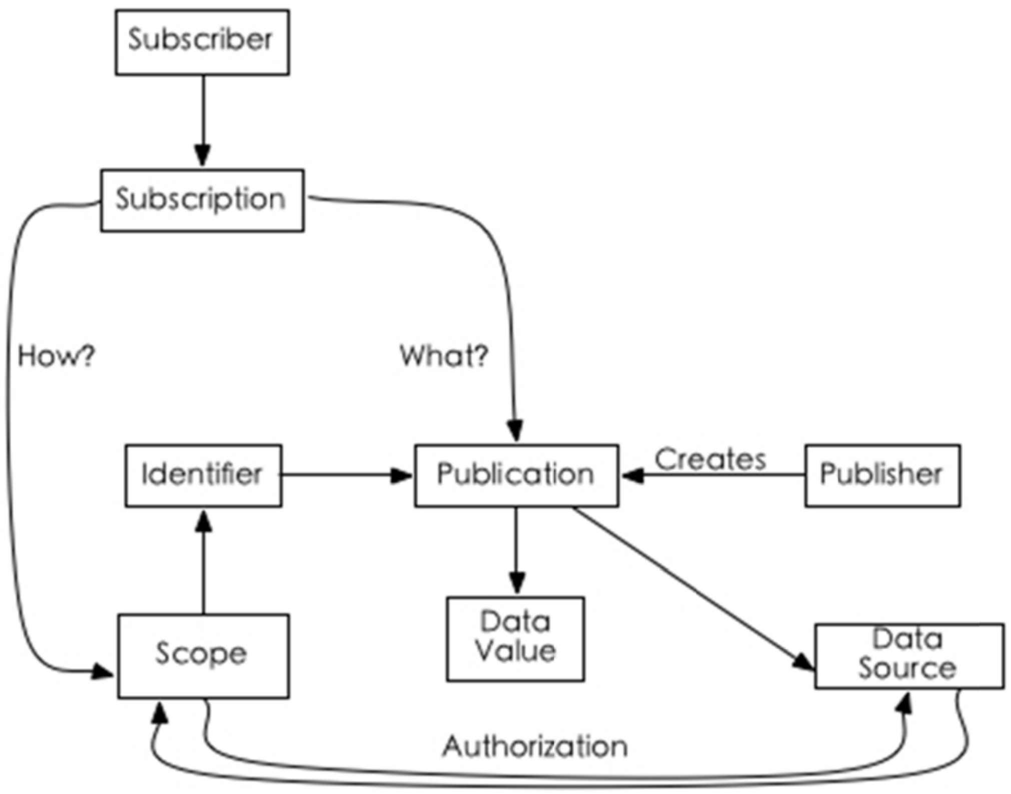


Figure 2-1: The basic relationship between publication, identifier, subscription, and scope. (Adapted from figure 1 of [9])

The PURSUIT continues the design and development of PSIRP project. It is an architecture design project that attempts to build solutions for a new form of Internetworking. The PURSUIT has improved objectives [11], which include providing an improved impedance match towards application-level concepts, providing tussles delineation of crucial functions, enabling optimization of sub-architectures, providing high performance and scaling to the needs of future Internet. The Blackadder is PURSUIT's new prototype implementation of an information-centric networking environment. It exports a pure publish/subscribe service model to application. The newest version that we currently use is supported on Linux, and provides Pub/sub API to developers. Chapter 5 will introduce the Blackadder prototype in detail.

3 PSIRP Concept and Architecture

In this chapter, we will introduce the basic architectural concepts and service model of PSIRP. The PSIRP network architecture consists of several entities, including data and metadata, identifiers, scoping information, subscribers and publishers, and domains.

Figure 3-1 shows the PSIRP component wheel and the relationship between identifiers and the functions. This pub/sub architecture has application programming interfaces (APIs) to ensure that different applications and networking functions can operate in it successfully. There are four classes of identifiers: application identifiers (AIDs), scope identifiers (SIDs), rendezvous identifiers (RIDs), and forwarding identifiers (FIDs). AIDs are resolved to RIDs and eventually to FIDs. The metadata includes scoping information and other important settings for the network elements. SIDs can be considered a subset of RIDs. After an item is published and subscribed to, the application can resolve an AID to a RID and transfers the item via the network. The metadata contains scoping information, which is associated with the RID. The system uses the scoping information to map each RID to FIDs, both within a domain and between domains. If this process is completed within a domain, then this operation is called **intra-domain routing and forwarding**, otherwise, it is called **inter-domain routing and forwarding**. Caching is a network function and can occur both on a local scale or all over the network [11].

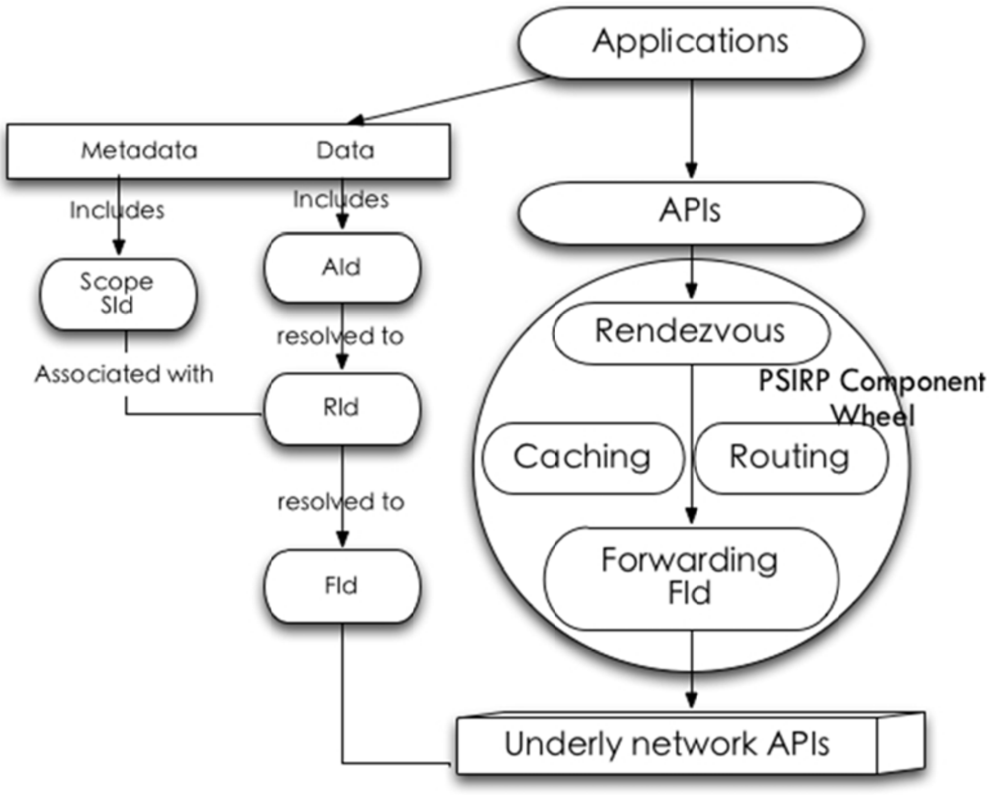


Figure 3-1: PSIRP Component Wheel and entities (Adapted from figures 7.1 and 7.2 of [11])

3.1 Data and metadata

Information is the key focus of PSIRP. Everything is information [9]. In the PSIRP architecture, the data is the center of attention. Every piece of information (item) in the information graph is related to some application task. A rendezvous identifies an item and the network delivers items to their destinations individually by utilizing different types of identifiers. However, each item is usually connected to other items on the application level. Publishers send their metadata to the rendezvous node. This metadata not only represents the relationship with other information, but is itself data with respect to the network level. Any item could be metadata for other items. On the network level, network metadata may represent the state of the network, encapsulated inside a communication header, or set as separate identifiers [11].

3.2 Scope

In the pub/sub information routing paradigm, developers can set information scoping mechanisms that limit the reachability of information with respect to a particular set of senders and receivers of a particular piece of information. Every item has a scope identifier (see section 3.3.3). The concept of scoping can be compared to topologies in the endpoint-centric IP world. During the publication operation, an explicitly identified scope is attached to a specific piece of information by its publisher. Using the scoping mechanism, publishers and subscribers can select a scope. Such a scope not only classifies the information items, but groups the interests as well. Subscriptions may be aggregated into more scalable scopes, which bring a key benefit to inter-domain network management. Scoping also allows separate publications to have shared control of their dissemination, as well as separate authentication. PSIRP is also able to aggregate operations on the scope, for instance, access control and other metadata-controlled operations; such as defining a scope-level caching policy [11]. Every item of information could belong to more than one scope. A video of a lecture can reside in college's academic scope and a user's personal scope. In pub/sub, information is structured as a directed acyclic graph. A scope may contain sub-scopes and information items, which means there are multiple levels of scopes. This brings some predictable benefits when a future network requires scalability and migration. Under a scope, everything is assigned a statistically unique ID. One or several paths starting from one or more graph's root can trace all scope and information items. Scopes can easily be built or re-constructed, removing specific parts from the scope, adding new sub-scopes, or information pieces. Also the information can easily be assigned to a new scope.

As we can see in Figure 3-2, universities or organization are responsible for different parts of projects, to be specific: the Pursuit and PSIRP projects. The sub-scope Deliverables is available to all these three scopes. Each scope has a governance policy attached, represented as metadata, which includes authentication information for potential receivers of information. Scopes provide social relations among elements, representing publisher, subscriber, and the information item itself.

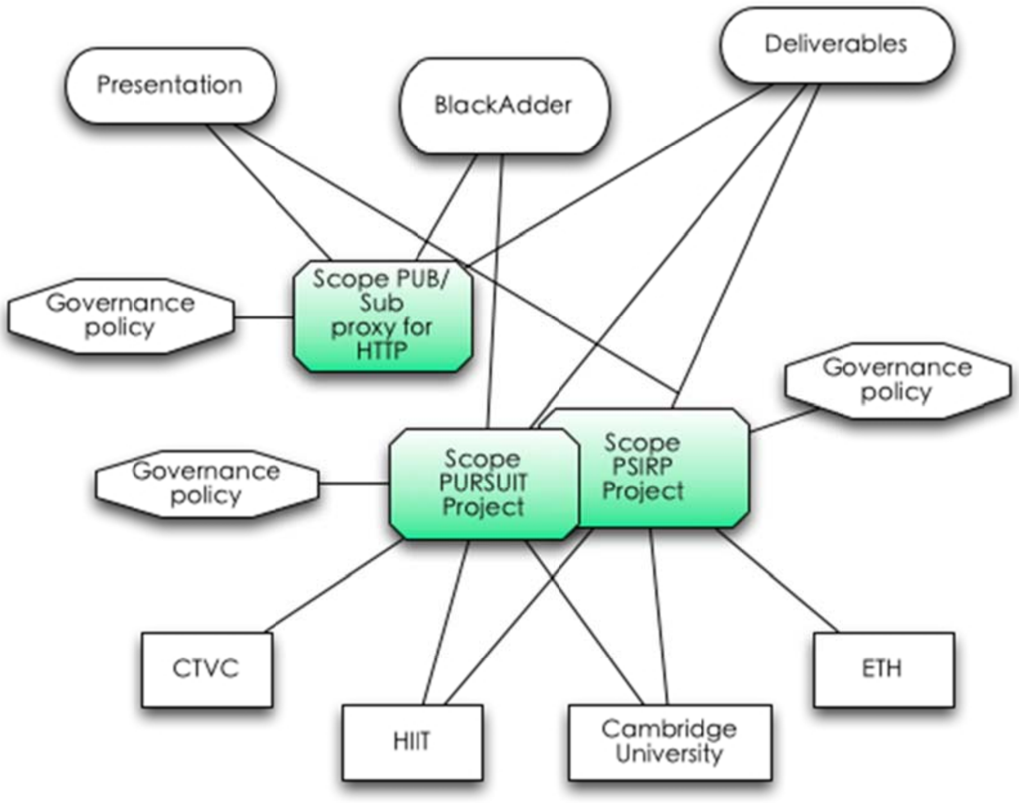


Figure 3-2: Example of PSIRP scope (Adapted from figure 6.1of [11])

3.3 Identifiers

In the pub/sub architecture, there are four types of identifiers. AIDs can be used to express and structure information on a high level. PSIRP utilizes self-certifying RIDs and SIDs to make the system more secure, and to enable nodes to independently verify the authenticity and integrity of information. A FID is based on a 256-bit BLOOM filter. The filter specifies the sets of links used for a path segment in the network. The FID encodes the delivery tree (i.e., the network path) and is encapsulated in the packet's header by the sender [9]. Each of these types of identifiers is described in a subsection below.

3.3.1 Application Identifiers (AIDs)

From the application's point of view, an identifier does **not** need to be unique, as an AID is only meaningful and can only be applied *within* a process. Applications can utilize many identifier concepts, such as flat or structured identifiers, and human-readable namespaces identifiers. Based on their own needs, AIDs are determined by application developers. In the PSIRP architecture, applications are expected to implement their own versions of identification and naming schemes. At the same time, other applications may rely on shared rules and services to collaborate within the network. An application needs to identify which network rendezvous identifiers are used *before* building a communication connection between sender and receiver. This action could be performed either by the application itself or by some service(s) in the network.

3.3.2 Rendezvous Identifiers (RIDs)

In order to communicate and connect different application identifiers, the network needs to use unique rendezvous identifiers (RIDs). An AID is not required to be universally unique, so it is impossible to directly use it for routing and forwarding data. In the PSIRP network, RIDs are used by rendezvous functions. When using the same semantics and RID, a publisher and subscriber can transmit information via a rendezvous node. Such an RID can be generated by any application executing at the sender or receiver and shared with other applications. Regarding the routing operations in the network, a RID may be utilized in the network authentication and access control process. Also, an RID is the bridge to connect higher-level identifiers (used in an application context) and lower layer identifiers (used in a network context). The network uses the same RID to establish communication among all participants, as well as when routing and forwarding requests.

3.3.3 Scope Identifiers (SIDs)

A scope identifier (SID) is attached to each rendezvous identifier to delimit the reachability of information. In the PSIRP architecture, SIDs is considered a specialized subclass of RIDs, which can be used by the rendezvous system to aggregate other RIDs. The pair (RID, SID) of an information item will determine the right set of FIDs for delivery of this data to a destination. One RID could be assigned to more than one SID. Every publication has its own separately controlled scope. As shown in the Figure 3-3, scope 0 is a root scope.The 'Pub' in the figure means publication. A publication can belong to more than one scope.

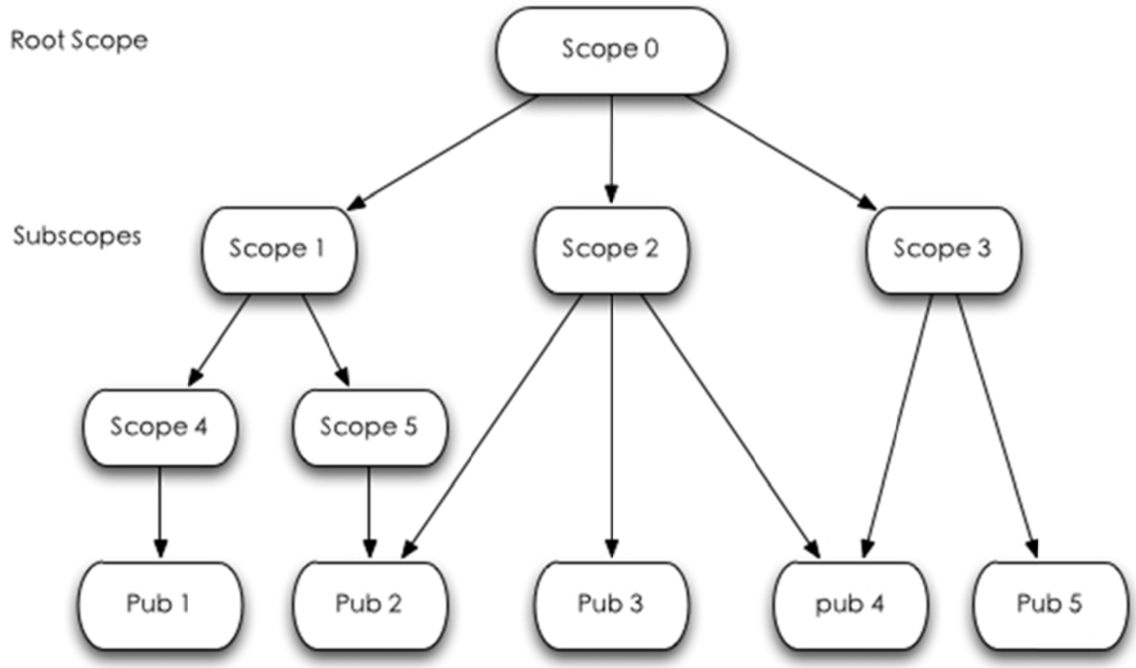


Figure 3-3 : An example of hierarchy information structure of PSIRP

3.3.4 Forwarding Identifiers (FIDs)

In the forwarding function of PSIRP, forwarding identifiers (FIDs) are utilized to accomplish data communication among publishers, subscribers, and system components. In order to send data from a publisher to the subscribers, the network has to resolve each pair of RID and SID into one or more FIDs. This function is achieved by the intra-domain and inter-domain rendezvous systems. The rendezvous systems play an intrinsic role in generating forwarding identifier resolution. The network can produce logical forwarding trees, which use the forwarding identifiers. [11]

3.4 PSIRP Components

This section will present some of the components of a PSIRP network, including the rendezvous system and the intra-domain & inter-domain routing and forwarding architecture.

3.4.1 Rendezvous

In an information-oriented internetworking architecture, the operation of finding and matching the correct information with a client is accomplished by the rendezvous function. A rendezvous determines the set of subscribers and publishers for a particular publication. To accomplish this requires specific underlying transport indicators to implement the data delivery. The rendezvous function takes place *between* network devices, which are called rendezvous points (RPs). RPs are logical meeting places within the pub/sub system. A publisher creates a scope of publication and sends this to the local RP node (in the Blackhawk and PURSUIT projects this is called a RZV node), and then the local RP node stores scope information. When a scope is published, the rendezvous system notifies all subscribers that have previously subscribed to this scope. A subscriber indicates an interest in this information item by sending metadata to the local RP node. If the requested scope does not exist, then the RP node will create a new scope. If publishers have already published (i.e., advertised) this item, then the RP node will match the item with the subscriber’s request. After that, the subscriber officially subscribes to the data from the publisher through the RP node. When the publisher receives a request, it starts to publish the data continuously. An example of such a message exchange process is shown in the Figure 3-4.

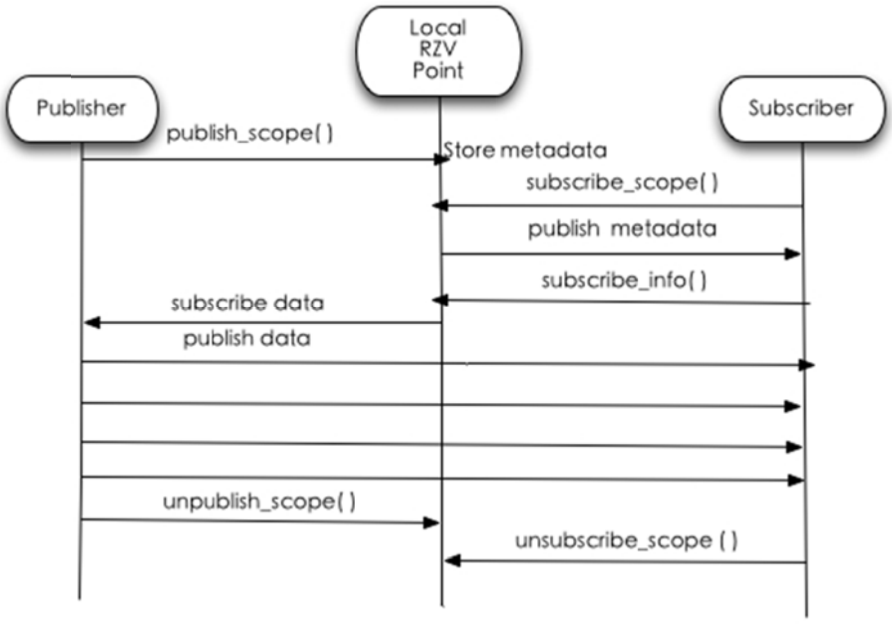


Figure 3-4: Example of message exchange process

The collection of RPs is called a rendezvous system in the PSIRP architecture. In the PURSUIT project, the developers explain three major rendezvous strategies: a node/link-local, a domain-local, and a global strategy for disseminating information.

The example, shown in Figure 3-4, is a scenario in which the rendezvous function is designed to be performed on a local scale, which means that all publications and subscriptions take place in the same node. In this case, local look-up tables could with low latency satisfy the queries and updates, instead of adding networking functions to the rendezvous process. Some helper functions related to topology management and network management in general take place in a link-local manner. To pair the publication and subscriber's needs, we can filter the irrelevant information on a local scale. In this project, we plan to set up a PSIRP overlay that includes several proxies. Each proxy acts as one element in a pub/sub network. Initially, we use the node/link –local strategy so that the operation of the rendezvous function is similar to a service discovery solution.

The domain-local and global rendezvous strategies are more complicated. For domain-local dissemination, there are two differences from node/link-local dissemination. The first difference is how to search for the appropriate RPs. The second difference is the process of forwarding publications and subscriptions between RPs. In a domain-local scenario, we assume that three kinds of elements exist in the network: forwarding nodes (simple, low complexity, packet switching devices), rendezvous nodes, and hosts [12]. Each forwarder is assigned a node identifier that is unique within the domain. By following the Line Speed Publish/Subscribe Inter-Networking (LIPSIN) mechanism[10], the forwarders compute the appropriate paths between all pairs of source/destination. Rendezvous nodes (RNs)[*] keep track of all active publications within a domain. Publishers send scope changes and advertisements to RNs, while the subscribers send their consumption requests to RNs. RNs match these requests with the advertisements. Flooding is

[*] An RN is a physical instance of a logical RP.

used by the forwarders to discover an RN’s location. The RNs advertise their location to the directly connected forwarders. These forwarders flood the RN’s information to other forwarders within the network. End-users send their publications and subscriptions to their attached forwarders, who will in turn transmit this information to the RNs. A rendezvous system usually has more than one RN. Every RN generates a metric list that shows the distances between forwarders by counting hops. When a new forwarder joins the network, it requests the list of available forwarders attached to RNs and sends its publications or subscription requests to the nearest one [11]. To achieve greater efficiency, RNs should be arranged around popular areas. Popular area means the place has high frequency of PSIRP requests and more active users. If RNs are installed in popular areas, it could reduce rendezvous delay [11].

3.4.2 Topology Management

One of the important components in the PSIRP architecture is the topology management function. This component is responsible for building a usable and efficient delivery graph for an established pub/sub relationship. While considering the different policies and network traffic conditions, the topology management system searches for available forwarding paths from publishers to subscribers, and then selects the most optimal one.

In the PSIRP project, the topology management functionality is divided into intra-domain and inter-domain mechanisms. Intra-domain topology management works within one administrative domain, its role is to discover topology information, computes and updates network states, and then forwards this information to other participated nodes. Topology management is associated with the rendezvous and forwarding functions. The PSIRP prototype allows client modules and server modules to coexist in the same node. The client module operates on each forwarding node. Its role is to discover local connectivity information. At the same time, the server module collects this information and maps them to compute a complete network topology. In addition, the server module computes the optimal forwarding paths, and then exchanges this information with forwarding nodes. This topology management implementation is similar to link state routing protocol link state advertisements (LSAs). More specifically, there are two scopes that exist in the process. One of them is the “Hello” scope, which is responsible for collecting “Hello” messages. Every node periodically publishes a “Hello message” including the node’s ID and basic information as an announcement of its existence to the “Hello” scope. The other scope is a link state advertisement scope. This scope collects information showing the set of neighboring nodes. The nodes that subscribe to the same scope listen to their neighbors’ advertisement messages and create a list of neighbors. When the topology changes, a new link state advertisement is published [13].

In the PURSUIT project, the developers aim to fully take advantage of the information-centric model, so they integrated the network attachment process with topology discovery. This strategy lets the scope structure represent the link structure of the local network. Basically, every node announces its existence by publishing a new scope, either a root scope or a sub-scope. The node’s ID determines the scope ID. All the announcement information can be spread using a broadcast strategy. In this way, nodes can communicate with other nodes whose location is within their transmission range. Every node that wants to join in the network’s activities has to subscribe to this strategy in order to get the information it wants. Eventually, the scopes not only identify the ID of the incoming node, but also provide location information and interconnectivity with other nodes.

3.4.3 Forwarding

Forwarding is one of the three most important main functions of the pub/sub model. We assume that our network does not rely on endpoint-based end-to-end addressing, such as an IP address. That requires the forwarding structure to be able to name information items. Senders will not able to send any data to clients who have not sent a subscription. This rule prevents DDoS attacks since every subscription has to follow the forwarding tree. It also requires that the forwarding tree is scalable and has access control. The FID is not restricted to one RID, which means that a particular FID can be used for different RIDs. When a new subscription is created, the existing transmission trees, identified by FIDs, can be re-used. This reduces the amount of network state. We can also divide the forwarding function into intra-domain and inter-domain forwarding.

In a domain, the link between two forwarding nodes has a link ID, which determines the packet forwarding. Each link between two forwarding nodes has two link IDs, one in each direction. The PSIRP structure combines two approaches together to transmit packets. The topology management system generates Bloom-filters-based FIDs and installs new state at the forwarding nodes. With link IDs and interconnectivity information, the topology management system creates a network graph. This network graph makes it possible to generate a forwarding tree for publications, as it has the locations of all the publishers *and* subscribers. Using this network graph, the topology management system creates a delivery tree for each publication. The delivery tree includes the links and nodes through which the information will pass. A Bloom filter encodes the entire set of link IDs as zFilters [10]. This filter is put into the packet's header. The rendezvous system considers all the participants (including subscriber and publishers) when the topology management system collects the network graph. Using this information, the zFilter defines a delivery tree. Additionally, policy restrictions and traffic engineering could also be considered when selecting the delivery tree. However, as the number of links included in the FID grows, Bloom filter based forwarding faces scalability issues because the proportion of falsely routed packets grows. At the same time, a simple Bloom filter based forwarding solution is subject to vulnerabilities, such as Bloom filter replay attacks, packet storms, forwarding loops, flow duplication, and injection attacks. More details about forwarding security problems and solutions are listed in the deliverable D2.2 of the PURSUIT project[10]. The developers in the PURSUIT project proposed an advanced forwarding security solution, called a Bloom filter-based Relay Architecture (BRA), which improves scalability in comparison with the former Bloom filter version. BRA divides the large-scale delivery tree into hierarchically arranged sub-trees [10]. Each sub-tree has its own Bloom filter that is used to forward a packet to the underlying sub-trees. The sub-trees are connected using relay nodes. These relay nodes deal with the forwarding information in the packet's header. The solution we described above is used within a single node. The next paragraph will describe how BRA works in a multi-area network.

If the network has several domains, we setup a rendezvous/topology node R in each domain. The node R has full topology information for the whole network. It is responsible for generating the delivery tree of its network and maintaining local subscriber information. The rendezvous nodes communicate with each other. Each R calculates a Bloom filter (BF) for the path from the rendezvous node (RN) to the receivers. The BF and the publication ID are sent to a new assigned RN. The receivers can either be a subscriber itself or other RNs in other areas. Rp is the closest rendezvous node to the publisher that stores publication information, while Rs is the closest R of the subscriber that maintains the subscription information as well. In a simple scenario, a client sends the subscription to Rs, the Rs checks if the subscription already exists in the local area. If not, then the Rs generates a delivery tree from the edge router to all subscribers in the local area, and then creates a Bloom filter. Finally, the subscription, the Bloom filter, and the number of passed links and subscribers are sent to R in the backbone. R’s job is to generate the multicast tree from the publisher to all the subscribers. If the number of links on a path is too great, then the R node will create a new RN in the backbone. Rp generates the delivery tree from the publisher to RN in the backbone with the BF created by R [11]. More examples can be found in [10].

Another solution developed to provide the forwarding function in pub/sub architecture is a multi-stage Bloom filter solution. To avoid a high false positive rate, this solution also uses a hierarchically organized delivery tree, but the delivery tree is arranged into several stages. In each forwarding stage, the individual Bloom filter identifiers are calculated in the same manner as in the LIPSIN [14] approach. This scheme utilizes a variable length FID, which reduces the overhead. However, the variable length identifier adds additional overhead and a multi-stage forwarding strategy is more complicated.

4 HTTP Proxy and Pub/sub HTTP proxy

The Hypertext Transfer Protocol (HTTP) has been used as an application-level protocol in the World Wide Web global information initiative since 1990 [17]. In our project, HTTP is used for communication between user agents and proxies or gateways to other Internet systems. These other systems may utilize SMTP, NNTP, and FTP for their communications.

4.1 Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) provides hypertext transfer from WWW servers to local browsers. HTTP not only ensures that the computer correctly transfers the hypertext documents, but also can determine which part of a document should be transferred, as well as which part of a webpage should be displayed first (for example, text takes precedence over graphics).

The HTTP protocol is a request/response protocol. The overall operation can be described as follows:

1. A client sets up a connection with the server.
2. The client sends a request to the server in the form of a request method, URL, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possibly a body containing content.
3. The server responds with a status line, followed by a MIME-like message containing server information, entity Meta information, and possible entity-body content [20].
4. The client receives the information returned by the server. If the client has no further requests, then the client and server disconnect. If errors occur during an operation, the error information will be returned to the client.

In the case of a web browser client, the process is now complete and the results are displayed to the user by the browser. This webpage may also have additional content which the user might request. In this case the user simply clicks on an object (with an associated URL) on a page with their mouse and waits for the new information to be displayed.

4.2 HTTP Proxy

An HTTP proxy is designed to proxy HTTP requests from clients who wish to obtain information. The proxy is a transit point between the user's browser and a web server. The proxy acts as an intermediary, simply passing the client's request on to the web server. When using a proxy the browser will not receive webpages or contents directly, but rather the client sends a request to a proxy server. The proxy gets the information and sends the requested information back to the client. The proxy server may also cache content (i.e., the results of earlier requests). If the same client or other clients requests the same object before the content expires in the cache, then the proxy directly transmits the cached data to the requesting client, rather than requesting the data from the web server. This function not only significantly improves browsing speed and efficiency, but accelerates the download and saves bandwidth (on the network path from the proxy to the actual web server). In many cases, the proxy server provides additional security – for example, hiding the identity of the requesting client or using a secure tunnel to get the information from the web server (even though the client has not explicitly ask to use a secure tunnel). By using a proxy the users can hide their real IP address from the web server, protecting them from a direct attack as the web server sees *only* the IP address of the proxy. The proxy may also perform filtering of requests or filtering content to prevent specific types of content from being accessed, to prevent clients from accessing certain web servers (and their associated web sites), or to prevent certain types of information from being sent by the client to the web server.

An HTTP proxy can be used to breakthrough IP access restrictions faced by clients in order to allow the clients to access websites outside a country (or company) or to enable access to some specific resources. For example, an FTP service could limit requests from clients to only those which are connected to the university's network (the proxy checks that the client's IP address is in the scope of allowed IP addresses), thus students could download and upload documents, share files, and access specific resources via the proxy server operated by the university. As another example, at KTH a proxy is used to enable students to access e-journals via the KTH Library's subscriptions to the journals, while ensuring that only those permitted to access the e-journal can make requests to the journal's web site. In this case only the web proxy needs be authorized to access the e-journal, rather than requiring that each student have their own subscription to access the e-journal.

4.3 HTTP Proxy Caching

A web cache reduces bandwidth usage, web server load, and the user's perceived delay by storing documents into temporary storage (a cache). Caching proxies are usually located near network gateways to reduce bandwidth consumption and decrease user-perceived response times. According to specific requirements or rules, the cache stores copies of content that is passing through it. If the requested object is in the cache, the proxy checks to see if the object is fresh enough to serve. If it is fresh, then the proxy serves it directly to the client. If the data in the cache may be stale, then the proxy connects to the origin server and checks if the object is still fresh. If it is, then the proxy immediately sends the cached copy to the client and updates the time that the object may be cached. If the object is not in the cache or if the server indicates that the cached copy is no longer valid, then the proxy obtains the object from the origin server. The object is passed to the client and stored in the proxy's local cache. Subsequent requests for the object can be served faster because the object can be retrieved directly from the cache.

HTTP defines three basic mechanisms for controlling caches: freshness, validation, and invalidation. Freshness allows a response to be used without rechecking it on the origin server. Both the server and the client side can control the degree of freshness. For example, the Max-age directive determines the lifetime of an object in the cache. Validation can be used to check if a cached response is still valid after it becomes stale. Invalidation is usually a side effect of another request that passes through the cache [18].

In order to evaluate the performance of a proxy cache, we generally consider the hit rate and the byte-hit rate. The hit rate indicates the percentage of all requests that can be satisfied by searching the cache for a copy of the requested object. The byte-hit rate is the percentage of all data that is transferred directly from the cache rather than from the origin server. If the requests are for small objects, it is easy to increase the hit rate by storing a larger number of small objects In contrast; caching a few large objects could increase the byte-hit rate. The proxy's cache has to rapidly make a decision about whether a requested object is cached to avoid increasing the latency of the response to s request.

4.4 Pub/Sub HTTP proxy towards the Internet and HTTP browsers

The PURSUIT developers have designed a Pub/Sub HTTP proxy that allows client to use a normal HTTP web browser to request data from the Internet. The subscribe-GET and subscribe-POST methods were implement with a publish/subscribe semantics network. On the client side the users use a standard browser, such as Google’s chrome, Microsoft’s Internet Explorer, Mozilla’s Firefox, or Apple’s Safari. The HTTP proxy connects to a Blackadder Information-Centric Network (ICN). This web proxy acts as a converter since it transforms normal HTTP GET requests to BlackAdder’s subscribe_GET format. In a BlackAdder subscribe_GET request, two identifiers are defined by using the Fowler/Noll/Vo (FNV) hash algorithm [19]. The input to this hash function are a domain related identifier and a subscription related identifier. The proxy forwards the BlackAdder request to the ICN network and subscribes to the domain’s scope and information items. The subscriber waits for a response or a time out. If the requested content is published before the time out, then the publisher will get a notification from the topology manager and it will publish the requested information to the subscriber. Since it is impossible for a publisher to publish all contents in the overlay in advance of requests, the developers created a mechanism that requires content providers to subscribe to a domain_request scope for the domain to which they wish to provide service. This enables the publishers to response to subscription requests. When the requested content does not exist within any scopes, then the subscriber publishes a request to this special scope to trigger a dynamic content domain publisher. After making this request the subscriber waits for a response from the network.

The proxy also can convert a pub/sub HTTP request to a standard HTTP request if the data has to be fetched from a remote HTTP server. To do so the proxy connects to the remote server by using a worker thread, setups a TCP connection, delivers the request and waits for a response. After receiving the content, the Pub/Sub HTTP proxy publishes the object to the BlackAdder ICN. As of the time of this writing, the development of this Pub/Sub HTTP proxy is still in progress. It is also expected that some caching schemes will be added in the Pub/Sub HTTP proxy. Further details about this Pub/Sub HTTP proxy can be found in [24].

Compared to this Pub/Sub HTTP proxy, the proxy we developed has a different structure and different functions. The PURSUIT HTTP proxy simply transforms a HTTP request to pub/sub request and forwards it to the BlackAdder Testbed ICN or reads the pub/sub requests and converts it to HTTP format in order to send it to a remote server. Both the BlackAdder ICN and PSIRP overlay have a publish/subscribe function. However, the former one always converts the format of the request, and then subscribes of the content from the pub/sub network. So the proxy needs to wait for an existing publisher to respond or time out. Our system allows a client to send both HTTP and PSIRP requests, because the proxy can identify the type of request and it has ability to use different threads to process the requests. Chapter 6 describes the details of the proxy operation process.

5 Publish / Subscribe Prototype Description

This section describes our use of the PURSUIT Publish/Subscribe prototype and the process of setting up this prototype in an environment for testing. The API for different languages for an information–centric application is introduced in section 5.2.

5.1 PURSUIT prototype

We decided to use Blackadder prototype v0.3 as our implementation of an information-centric networking environment. Compared to the earlier Blackhawk prototype which only runs on FreeBSD, Blackadder is supported on Linux and FreeBSD. FreeBSD is widely used by computer science researchers. However, Linux offers some tools and drivers which are not supported by other operating systems. In this project, we used Linux as our operating system since it was easier to set up and there were a number of distributions to choose from. Blackadder has a C++ library for developing applications and also provides a wrapper for this library for C, Java, Python, and Ruby. We used Java to program our proxy. Blackadder implements the core functions of rendezvous, topology management, and forwarding. Moreover, Blackadder provides four different dissemination strategies within the PURSUIT functional model.

5.2 API Description

Blackadder communicates with applications through a Netlink socket [20]. First, the application opens a Netlink socket and transfers data buffers to Blackadder. All of these data buffers are compliant with a format which Blackadder recognizes as publish/subscribe requests. Then, Blackadder establishes information-related events and sends results back to the applications through a Netlink socket.

Blackadder's C++ API includes two singleton classes. The first class, Blackadder, handles publish/subscribe requests in a blocking fashion. The second class, NB_Blackadder, processes publish/subscribe requests in a non-blocking fashion [21]. In the former class all the requests are sent to the networking stack before each method unblocks in the application. For the second class, a selector and a worker thread are created at the same time. When applications receive publish/subscribe requests, they forward the requests to a queue.

Here, we will briefly describe Blackadder's C++ API, the SWIG-based Java API, and a non-SWIG binding. More details can be found in the API document in the PURSUIT prototype description [21]. This API is an improvement over the earlier Blackhawk prototype. As noted earlier, Blackadder provides language bindings for Java. These bindings are generated with Simplified Wrapper and Interface Generator (SWIG). These bindings wrap some functions in the C++ API so they can be used in other "higher level" languages. Hence, we can install SWIG bindings for Python, Ruby, and Java - after the installation of Blackadder's C++ API library, depending upon which language(s) we wish to use. One can also use the independent Java Wrapper, which is a non-SWIG Java binding for Blackadder. This later method has some classes that can offer an object-oriented abstraction to the network functions. Additionally, this Java wrapper reduces the number of buffer copies between the Blackadder library and the JVM, thus providing higher performance. For all of these reasons we choose the independent Java wrapper to develop our proxy. The next subsection will introduce both the C++ and Java API details that we thought might be useful in our project.

All the basic entities are included in the classes of the eu.package.core package. When an application wants to communicate with other elements in the ICN, it needs to create an instance of a BlackAdderClient object. This BlackAdderClient instance provides all the methods for publishing and subscribing to information.

5.2.1 Pub/Sub Methods

5.2.1.1 Publishing Scopes

In the pub/sub architecture, the rendezvous system maintains an information graph. Once a publisher creates a scope by sending a publish_scope () request, the rendezvous system will send notifications to those subscribers who have already subscribed to this scope. The publisher can define the information's visibility scale: within its local host (node_local), its host and a physical neighbor (link_local), or within the domain (domain_local). Both scopes and information items can belong to multiple paths in the information graph, which means that they can be identified by multiple information IDs.

The relevant methods are:

C++
```
void publish_scope (const string &id, const string &prefix_id,
unsigned char strategy, void *str_opt, unsigned int
str_opt_len);
```

Java
```
publish_scope (byte [] id, byte [] prefix_id, int strategy,
byte [] data)
publishScope ( )
```

5.2.1.2 Advertising Information Items

Information items could reside under more than one scope. Each information item has to be advertised under one or more scopes. In the DOMAIN_LOCAL dissemination strategy, depending on the existence of subscribers in the information graph, the rendezvous node may initiate the rendezvous process during the publication and subscription matching process. The rendezvous node could also ask the topology manager for topology formation. After receiving a request, the topology manager will notify the publishers to publish the data for the specific information item.

The relevant methods are:

C++
```
void publish info (const string &id, const string &prefix id,
unsigned char strategy, void *str_opt, unsigned int str_opt_len);
```

Java
```
publish_info (byte [] id, byte [] prefix_id, int strategy, byte []
data)
publishInfo ( )
```

5.2.1.3 Subscribing to Scopes

A client can subscribe to a scope. If a required scope does not exist, the rendezvous node will build a new one. When a subscriber subscribes to a scope, it means that the subscriber is interested in all the scopes (not the whole sub-graph) and information items under that scope. The scopes will ask the topology manager for a LIPSIN identifier. A series of LIPSIN identifiers are able to direct the information items from publishers to the subscribers.

The relevant methods are:

```
C++   void subscribe_scope (const string &id, const string
      &prefix id, unsigned char strategy, void *str opt, unsigned int
      str_opt_len);
Java  subscribe_scope (byte [] id, byte [] prefix_id, int strategy,
      byte [] data)
      subscribeScope ( )
```

5.2.1.4 Subscribing to Information Items

When a client wants to obtain an information item, the rendezvous node will match the previously advertised item to this subscription. Then, the rendezvous node publishes a request to the topology manager to generate LIPSIN identifiers from every available publisher to subscribers. The topology management system selects the best paths from the publishers to the subscribers and builds multicast trees.

The relevant methods are:

```
C++   void subscribe info (const string &id, const string &prefix id,
      unsigned char strategy, void *str_opt, unsigned int
      str_opt_len);
Java  subscribe_info (byte [] id, byte [] prefix_id, int strategy,
      byte [] data)
      subscribeInfo ( )
```

5.2.1.5 Unpublishing Scopes

An existing scope can be deleted from the information graph. The subscribers of the father scopes are notified about the unpublishing event. If the scope was published under multiple scopes, then the unpublishing action only deletes the specified branch.

The relevant methods are:

```
C++   void unpublish_scope (const string &id, const string
      &prefix id, unsigned char strategy, void *str opt, unsigned int
      str_opt_len);
Java  unpublish_scope (byte [] id, byte [] prefix_id, int strategy,
      byte [] data)
      unpublishScope ( )
```

5.2.1.6 Unpublishing Information Items

The publisher can delete a specific information item from the information graph. If this publisher is the only one concerned with the information item, then this item will also be deleted from the rendezvous system after an unpublishing request. However, if there are other publishers who have published this item, then the topology manager will update the LIPSIN identifiers. The relevant methods are:

C++
```
void unpublish_info (const string &id, const string &prefix id,
unsigned char strategy, void *str_opt, unsigned int
str_opt_len);
```

Java
```
unpublish_info (byte [] id, byte [] prefix_id, int strategy,
byte [] data)
unpublishInfo ( )
```

5.2.1.7 Unsubscribing from Scopes

When a subscriber wants to unsubscribe from a scope, it will be removed from the list of subscribers of this scope. The relevant methods are:

C++
```
void unsubscribe_scope (const string &id, const string
&prefix id, unsigned char strategy, void *str opt, unsigned int
str_opt_len);
```

Java
```
unsubscribe_scope (byte [] id, byte [] prefix id, int strategy,
byte [] data)
unsubscribeScope ( )
```

5.2.1.8 Unsubscribing from Information Items

After the subscriber send unsubscribing requests for a specific information items, the rendezvous node will check the information graph to see if there still are other publishers and subscribers. If nothing is left in the information graph, this information item will be deleted. Otherwise, the forwarding identifiers will be updated. The relevant methods are:

C++
```
void unsubscribe_info (const string &id, const string
&prefix id, unsigned char strategy, void *str opt, unsigned int
str_opt_len);
```

Java
```
unsubscribe_info (byte [] id, byte [] prefix_id, int strategy,
byte [] data)
unsubscribeInfo ( )
```

5.2.1.9 Publishing Data

When the matching process is finished, the topology manager sends FIDs to Blackadder. Blackadder assigns the FIDs to the information items. The relevant methods are:

C++
```
void publish_data (const string &id, unsigned char strategy,
void *str_opt, unsigned int str_opt_len, void *data, unsigned
int data_len);
```

Java
```
publish data (byte [] id, byte [] prefix id, int strategy, byte
[] data)
publishData ( )
```

Different dissemination strategies are defined by the Strategy enum in the eu.package.core package. Specifically, NODE, LINK_LOCAL, DOMAIN_LOCAL, IMPLICIT_RENDEZVOUS, and BROADCAST_IF strategies exist. We plan to initially utilize the NODE strategy in our prototype. Subsequently the LINK_LOCAL and DOMAIN_LOCAL strategies could be utilized in an experiment.

5.3 Blackadder Events

In the Independent Java Wrapper for Blackadder, the Event class expresses incoming notifications. This method will be blocked until the next message comes from the ICN. There are several types of events. Here, we will briefly describe their role in the process of receiving notifications from the ICN.

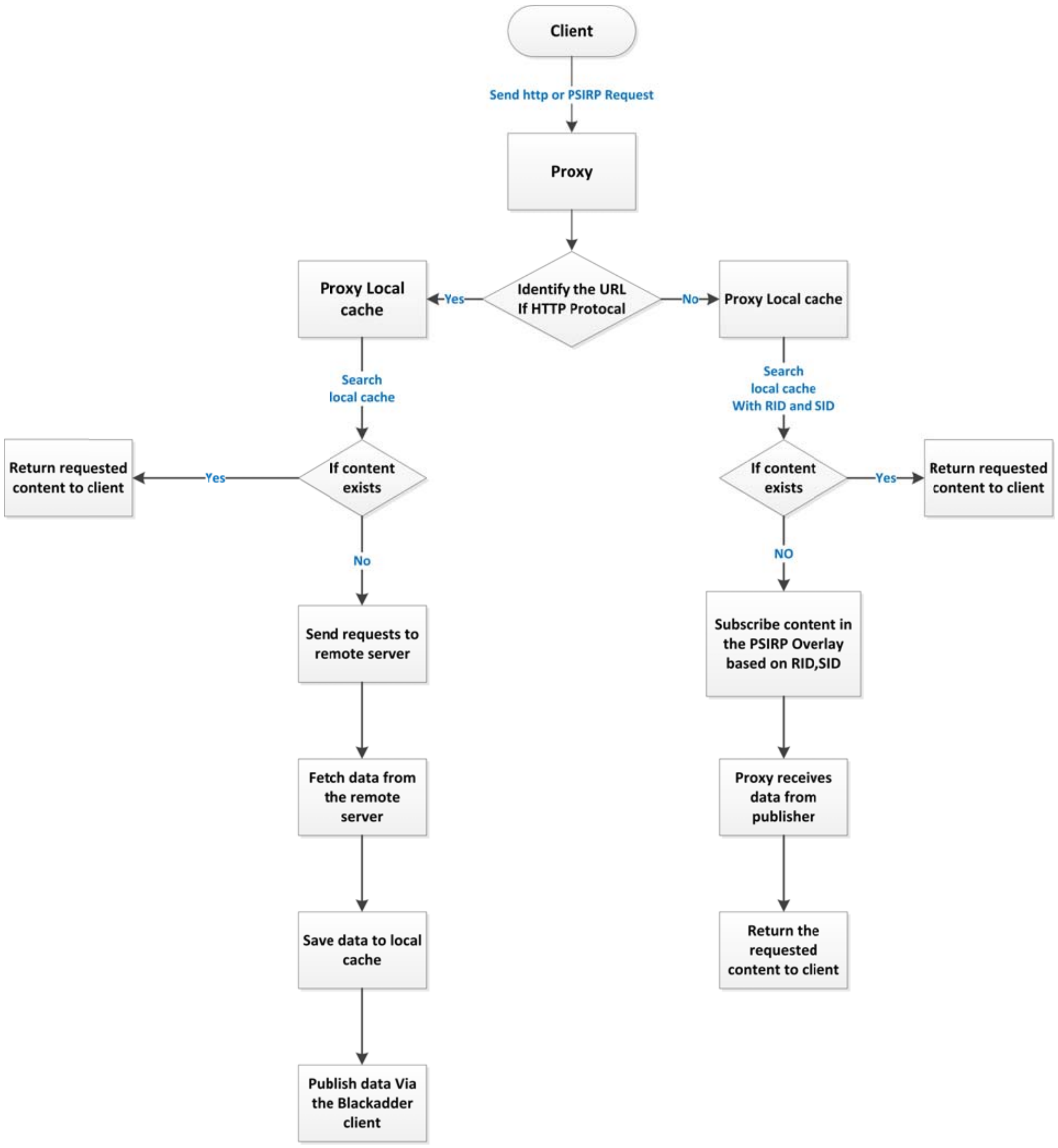
START_PUBLISH	When the forwarding identifier is available for an information item, Blackadder will send a START_PUBLISH event to an application. This event contains the information ID, which identifies the information item to be published.
STOP_PUBLISH	We assume that several subscribers initially subscribe to one information item. This information item has a FID. Later the subscribers decide to unsubscribe to this content one by one. Blackadder will send a STOP_PUBLISH event to an application after the last subscriber ends their interested in this information item.
SCOPE_PUBLISH	The SCOPE_PUBLISH event is a notification for the subscriber who previously subscribed to this scope. When a new scope or sub-scope is created under the subscribed scope, the subscriber will receive this type of notification. Additionally, the subscriber may also get such events if there are sub-scopes existing under the subscribed scope.
DATA	Applications publish data through DATA events. The information IDs are included in this event.

6 System Design of our prototype

Based on the Blackadder prototype and the requirements of pub/sub proxy for HTTP, we designed a program to realize our own prototype. This chapter explains the system design of our program which implements a pub/sub proxy for HTTP. We will introduce our program's work flow diagram and Class diagram. A Firefox browser plugin is also presented. The chapter ends with a discussion of the kinds of information that the system can transfer.

6.1 Proxy operation process

As we can see in Figure 6-1, the proxy acts as intermediary between the clients and the PSIRP overlay or a remote HTTP server. Each such proxy is an element in the PSIRP overlay. This overlay is an instance of a pub/sub architecture. Initially, a client utilizes the Firefox browser to send a content request to the proxy. In our test case, we used a command line interface to send our requests. The request could use either a PSIRP URL or HTTP URL. The proxy recognizes one these two types of URLs, then utilizes one of the two socket servers to process the request. If the URL begins with "http://", then the proxy will act as a normal HTTP proxy, thus it will first check its local cache. If the content is available in the cache, fresh, and valid, then the proxy returns the data directly to the client. Otherwise, it forwards the request to the origin web server. The origin web server returns the requested content to proxy. The proxy decides whether the content is cacheable or not. If the content is cacheable, then the content is saved in the proxy's cache and the proxy will publish this content to the rest of the pub/sub overlay. As a result the members of the overlay will learn what resources are stored in this proxy's cache. Finally, the proxy forwards the content to the client. If the URL begins with "psirp://", then this URL contains the RIDs and SIDs of the information item. When the proxy receives a "psirp://" URL, it will look for the requested content in its local cache. We utilize the RID and SID as a key to the local cache. If the requested content is in the proxy's cache, then the proxy returns the content to client, otherwise the proxy needs to subscribe to this information in the pub/sub overlay based on the RID and SID. After the content is delivered to the client, the client renders the content through the Firefox browser or other media player.



Client
Send http or PSIRP Request
Proxy
Identify the URL If HTTP Protocal
Yes
No
Proxy Local cache
Proxy Local cache
Search local cache
Search local cache With RID and SID
If content exists
If content exists
Yes
Yes
Return requested content to client
Return requested content to client
No
NO
Send requests to remote server
Subscribe content in the PSIRP Overlay based on RID,SID
Fetch data from the remote server
Proxy receives data from publisher
Save data to local cache
Return the requested content to client
Publish data Via the Blackadder client

Figure 6-1: The operation process of the pub/sub proxy for HTTP

6.2 Program work flow

We developed a program that works as a proxy server to handle both HTTP and PSIRP requests. Figure 6-2 shows how the system processes requests. Initially, we bootstrapped the system as two socket servers: a HTTP Socket Server and PSIRP Socket Server. This socket server is responsible for listening to a specific port on the host, TCP port 9800 and 9801 in our case.

Multiple threads are utilized to process the different tasks in parallel. The following subsections describe the HTTP socket server and PSIRP Socket server in detail.

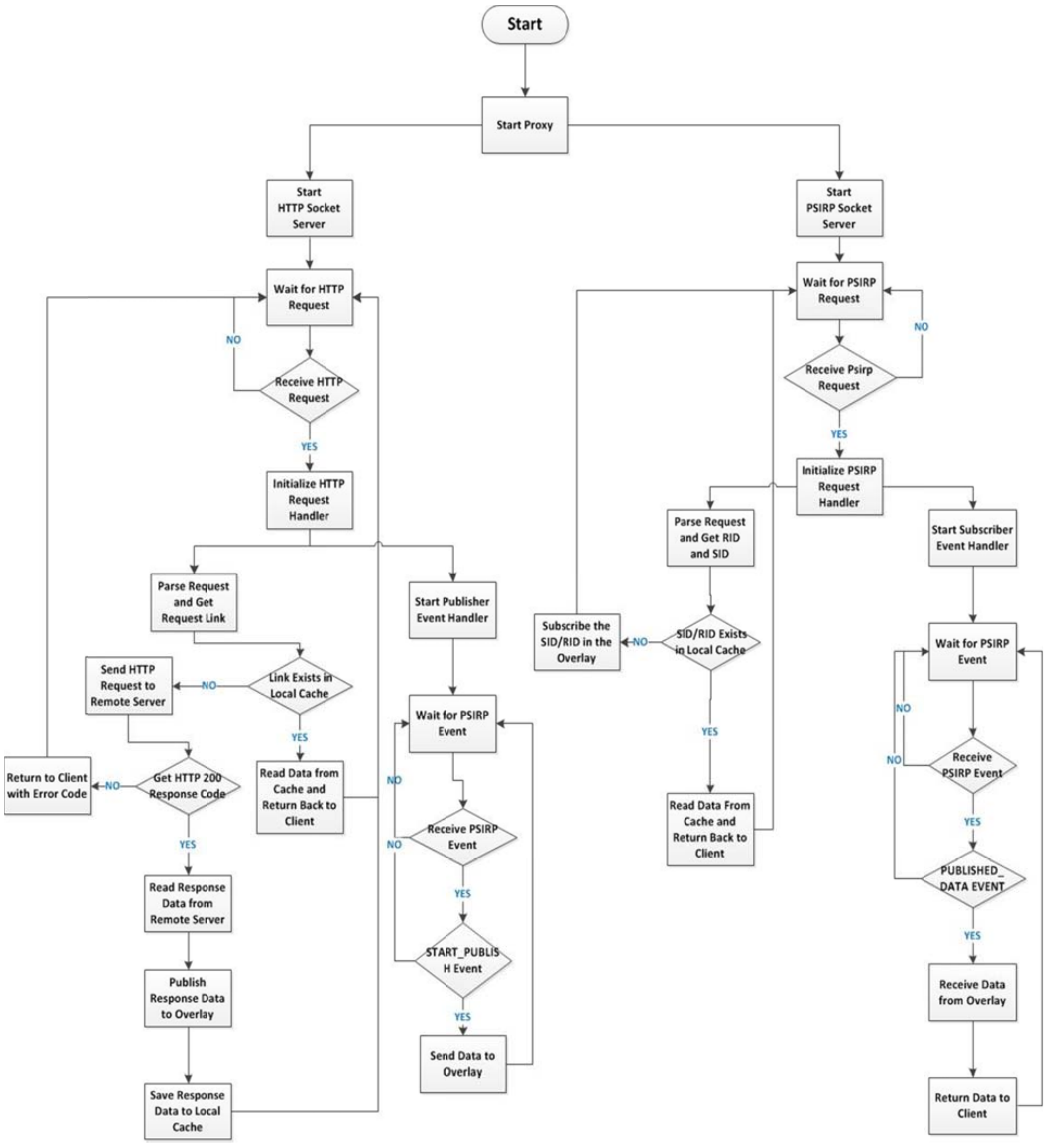

Start
Start Proxy
Start HTTP Socket Server
Start PSIRP Socket Server
Wait for HTTP Request
NO
Receive HTTP Request
YES
Initialize HTTP Request Handler
Parse Request and Get Request Link
Link Exists in Local Cache
NO
YES
Send HTTP Request to Remote Server
Read Data from Cache and Return Back to Client
Get HTTP 200 Response Code
NO
YES
Return to Client with Error Code
Read Response Data from Remote Server
Publish Response Data to Overlay
Save Response Data to Local Cache
Start Publisher Event Handler
Wait for PSIRP Event
Receive PSIRP Event
NO
YES
START_PUBLISH Event
NO
YES
Send Data to Overlay
Wait for PSIRP Request
Receive Psirp Request
NO
YES
Initialize PSIRP Request Handler
Parse Request and Get RID and SID
SID/RID Exists in Local Cache
NO
YES
Subscribe the SID/RID in the Overlay
Read Data From Cache and Return Back to Client
Start Subscriber Event Handler
Wait for PSIRP Event
Receive PSIRP Event
NO
YES
PUBLISHED_DATA EVENT
NO
YES
Receive Data from Overlay
Return Data to Client

Figure 6-2: Program work flow diagram of pub/sub proxy for HTTP.

6.2.1 HTTP Socket Server

After the HTTP socket server start running, it listens to TCP port number 9800 and waits for a HTTP request. Once a HTTP request arrives at the proxy, it initializes a HTTP request handler, which starts two sub-processes. The first sub-process performs the normal HTTP functions. It parses the request and gets the requested URL. If the content associated with this URL already exists in local cache, then the proxy directly reads the data from cache and returns this content to client. Otherwise, the proxy sends a HTTP request to the remote server and waits for a response. An HTTP response code 200 means the requested content can be successfully fetched from the HTTP server. Next, the proxy reads the response data from the remote server. Then, it publishes the content to the PSIRP overlay and saves the content in the local cache, if the content is cacheable. The final step is to deliver the requested content to the client.

The other sub process implements the publish/subscribe function of our proxy. The proxy starts a Publisher Event Handler, which is responsible for dealing with a PSIRP event. Once the BlackAdder event arrives at the handler, it activates the START_PUBLISH event. Next the proxy publishes the data to the PSIRP overlay. Other proxies will learn that this item is available from this proxy.

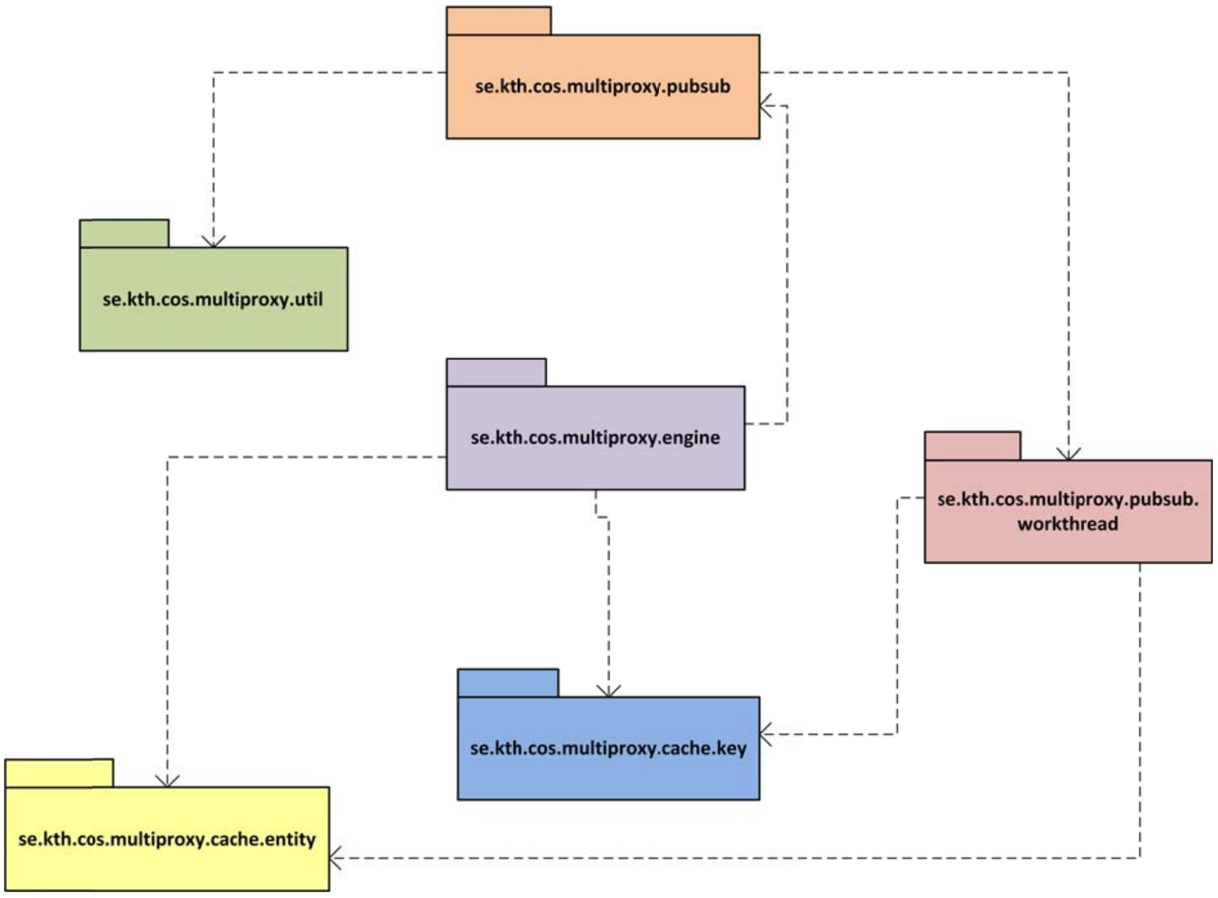
6.2.2 PSIRP Socket Server

Once the PSIRP socket server is activated by the main process, it will wait for a PSIRP request. Our program provides the catalog function to clients. The clients could request the catalog of existing PSIRP contents in the PSIRP overlay. They can choose the information they want and send a PSIRP URL to the proxy. When the proxy receives a PSIRP request, it initializes the PSIRP Request Handler. The PSIRP Request Handler starts two threads. The first parses the request and extracts the RID and SID of the content. Next, the proxy searches its local cache based on this SID and RID. If the requested content already exists in the cache, then the proxy will read this data and then send it to client. Otherwise, the proxy will subscribe for the content with this SID and RID in the PSIRP overlay. At the same time, the PSIRP Request Handler starts a Subscriber Event Handler, which should wait for a response from the overlay. After this handler receives a BlackAdder Event, it activates a PUBLISHED_DATA Event. The next step is to receive the data from Overlay. Eventually, the proxy returns the data to client.

6.3 Java Package Diagram

Figure 6-3 shows the Java package diagram of the proposed design. In order to make the code and system logic clean and well organized, we divided the design into six packages and each package includes classes which have similar functions.

The multiproxy.engine package contains a socket server class and a class that can control the work flow of the proposed system. All of the classes that are related to publish and subscribe activity are placed into the multiproxy.pubsub package. The classes in the multiproxy.pubsub.workthread package are responsible for data exchange. In the pub/subarchitecture, information recognition and forwarding is based on identifiers. We organize the utility classes together as a multiproxy.util package. For instance, there is a class that can generate RIDs and SIDs. The multiproxy.cache.key package and the multiproxy.cache.entity package are utilized to implement the local cache functionality. The relationships among these packages are shown in the Figure 6-3.

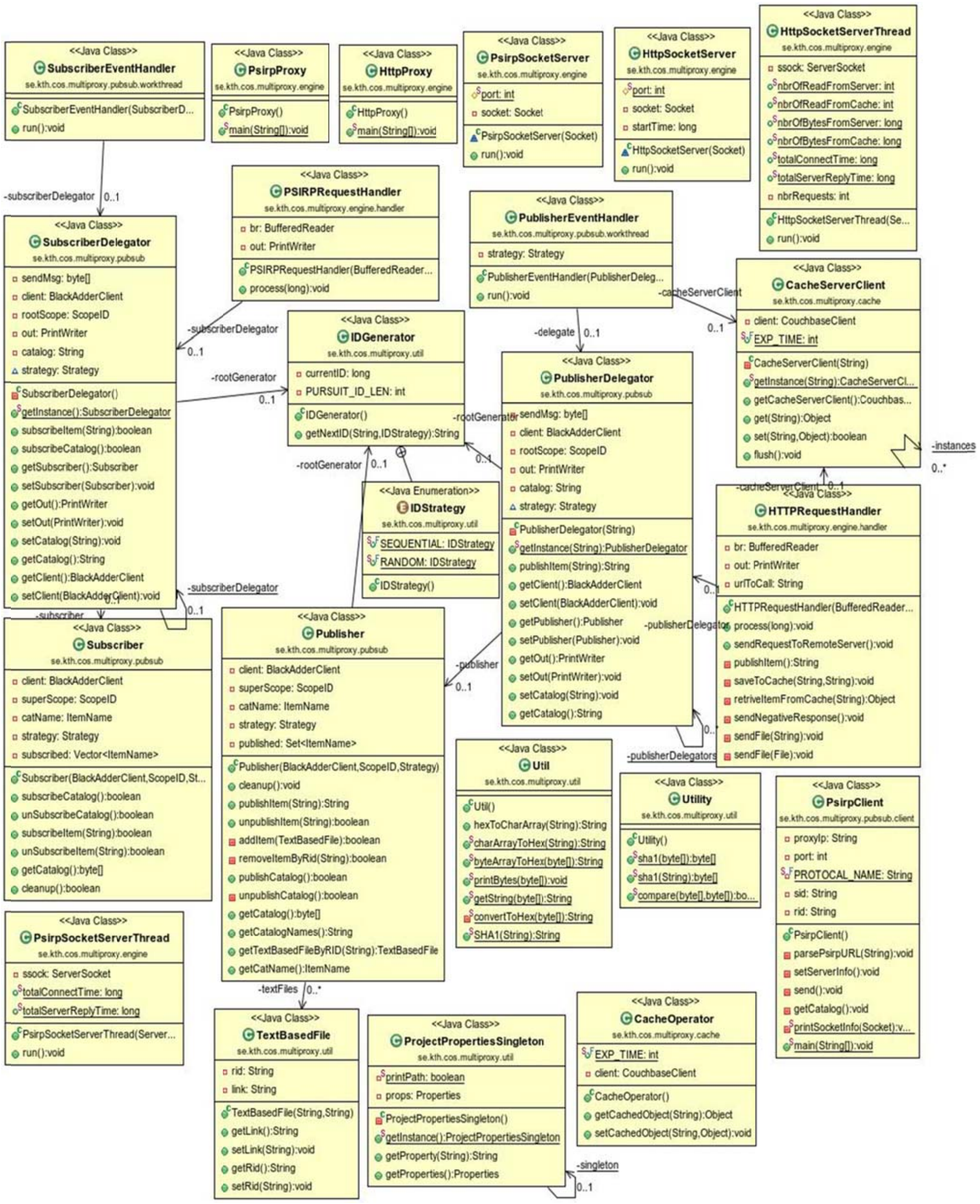


se.kth.cos.multiproxy.pubsub
se.kth.cos.multiproxy.util
se.kth.cos.multiproxy.engine
se.kth.cos.multiproxy.pubsub.
workthread
se.kth.cos.multiproxy.cache.key
se.kth.cos.multiproxy.cache.entity

Figure 6-3: The package diagram of proxy program

6.4 Class Diagram

Figure 6-4 shows the class diagram of the proposed system. It illustrates the relationships among the classes and lists the main methods for each of the core classes.



<<Java Class>> SubscriberEventHandler se.kth.cos.multiproxy.pubsub.workthread
SubscriberEventHandler(SubscriberD...
run():void
<<Java Class>> PsirpProxy se.kth.cos.multiproxy.engine
PsirpProxy()
main(String[]):void
<<Java Class>> HttpProxy se.kth.cos.multiproxy.engine
HttpProxy()
main(String[]):void
<<Java Class>> PsirpSocketServer se.kth.cos.multiproxy.engine
port: int
socket: Socket
PsirpSocketServer(Socket)
run():void
<<Java Class>> HttpSocketServer se.kth.cos.multiproxy.engine
port: int
socket: Socket
startTime: long
HttpSocketServer(Socket)
run():void
<<Java Class>> HttpSocketServerThread se.kth.cos.multiproxy.engine
ssock: ServerSocket
nbrOfReadFromServer: int
nbrOfReadFromCache: int
nbrOfBytesFromServer: long
nbrOfBytesFromCache: long
totalConnectTime: long
totalServerReplyTime: long
nbrRequests: int
HttpSocketServerThread(Se...
run():void
-subscriberDelegator
0..1
<<Java Class>> PSIRPRequestHandler se.kth.cos.multiproxy.engine.handler
br: BufferedReader
out: PrintWriter
PSIRPRequestHandler(BufferedReader...
process(long):void
<<Java Class>> PublisherEventHandler se.kth.cos.multiproxy.pubsub.workthread
strategy: Strategy
PublisherEventHandler(PublisherDeleg...
run():void
-cacheServerClient
0..1
<<Java Class>> CacheServerClient se.kth.cos.multiproxy.cache
client: CouchbaseClient
EXP_TIME: int
CacheServerClient(String)
getInstance(String):CacheServerCl...
getCacheServerClient():Couchbas...
get(String):Object
set(String,Object):boolean
flush():void
-instances
0..*
<<Java Class>> SubscriberDelegator se.kth.cos.multiproxy.pubsub
sendMsg: byte[]
client: BlackAdderClient
rootScope: ScopeID
out: PrintWriter
catalog: String
strategy: Strategy
SubscriberDelegator()
getInstance():SubscriberDelegator
subscribeItem(String):boolean
subscribeCatalog():boolean
getSubscriber():Subscriber
setSubscriber(Subscriber):void
getOut():PrintWriter
setOut(PrintWriter):void
setCatalog(String):void
getCatalog():String
getClient():BlackAdderClient
setClient(BlackAdderClient):void
-subscriberDelegator
0..1
-rootGenerator
0..1
<<Java Class>> IDGenerator se.kth.cos.multiproxy.util
currentID: long
PURSUIT_ID_LEN: int
IDGenerator()
getNextID(String,IDStrategy):String
-delegate
0..1
-rootGenerator
0..1
<<Java Class>> PublisherDelegator se.kth.cos.multiproxy.pubsub
sendMsg: byte[]
client: BlackAdderClient
rootScope: ScopeID
out: PrintWriter
catalog: String
strategy: Strategy
PublisherDelegator(String)
getInstance(String):PublisherDelegator
publishItem(String):String
getClient():BlackAdderClient
setClient(BlackAdderClient):void
getPublisher():Publisher
setPublisher(Publisher):void
getOut():PrintWriter
setOut(PrintWriter):void
setCatalog(String):void
getCatalog():String
-rootGenerator
0..1
<<Java Enumeration>> IDStrategy se.kth.cos.multiproxy.util
SEQUENTIAL: IDStrategy
RANDOM: IDStrategy
IDStrategy()
-cacheServerClient
0..1
<<Java Class>> HTTPRequestHandler se.kth.cos.multiproxy.engine.handler
br: BufferedReader
out: PrintWriter
urlToCall: String
HTTPRequestHandler(BufferedReader...
process(long):void
sendRequestToRemoteServer():void
publishItem():String
saveToCache(String,String):void
retrieveItemFromCache(String):Object
sendNegativeResponse():void
sendFile(String):void
sendFile(File):void
-subscriberDelegator
0..1
-subscriber
0..1
<<Java Class>> Subscriber se.kth.cos.multiproxy.pubsub
client: BlackAdderClient
superScope: ScopeID
catName: ItemName
strategy: Strategy
subscribed: Vector<ItemName>
Subscriber(BlackAdderClient,ScopeID,St...
subscribeCatalog():boolean
unSubscribeCatalog():boolean
subscribeItem(String):boolean
unSubscribeItem(String):boolean
getCatalog():byte[]
cleanup():boolean
<<Java Class>> Publisher se.kth.cos.multiproxy.pubsub
client: BlackAdderClient
superScope: ScopeID
catName: ItemName
strategy: Strategy
published: Set<ItemName>
Publisher(BlackAdderClient,ScopeID,Strategy)
cleanup():void
publishItem(String):String
unpublishItem(String):boolean
addItem(TextBasedFile):boolean
removeItemByRid(String):boolean
publishCatalog():boolean
unpublishCatalog():boolean
getCatalog():byte[]
getCatalogNames():String
getTextBasedFileByRID(String):TextBasedFile
getCatName():ItemName
-publisher
0..1
-publisherDelegator
0..1
-publisherDelegators
<<Java Class>> Util se.kth.cos.multiproxy.util
Util()
hexToCharArray(String):String
charArrayToHex(String):String
byteArrayToHex(byte[]):String
printBytes(byte[]):void
getString(byte[]):String
convertToHex(byte[]):String
SHA1(String):String
<<Java Class>> Utility se.kth.cos.multiproxy.util
Utility()
sha1(byte[]):byte[]
sha1(String):byte[]
compare(byte[],byte[]):bo...
<<Java Class>> PsirpClient se.kth.cos.multiproxy.pubsub.client
proxyIp: String
port: int
PROTOCAL_NAME: String
sid: String
rid: String
PsirpClient()
parsePsirpURL(String):void
setServerInfo():void
send():void
getCatalog():void
printSocketInfo(Socket):v...
main(String[]):void
<<Java Class>> PsirpSocketServerThread se.kth.cos.multiproxy.engine
ssock: ServerSocket
totalConnectTime: long
totalServerReplyTime: long
PsirpSocketServerThread(Server...
run():void
-textFiles
0..*
<<Java Class>> TextBasedFile se.kth.cos.multiproxy.util
rid: String
link: String
TextBasedFile(String,String)
getLink():String
setLink(String):void
getRid():String
setRid(String):void
<<Java Class>> ProjectPropertiesSingleton se.kth.cos.multiproxy.util
printPath: boolean
props: Properties
ProjectPropertiesSingleton()
getInstance():ProjectPropertiesSingleton
getProperty(String):String
getProperties():Properties
-singleton
0..1
<<Java Class>> CacheOperator se.kth.cos.multiproxy.cache
EXP_TIME: int
client: CouchbaseClient
CacheOperator()
getCachedObject(String):Object
setCachedObject(String,Object):void

Figure 6-4: The class diagram of the system

6.5 Firefox plugin

Currently, if a client wants to establish a long-term interest in an item of content, it has to periodically poll the web server to check if there is a new version of this content. This occurs because the rendezvous between the client and server is instantaneous, not permanent. However, both the web server and client could benefit from a long-term association. That is one reason why a pub/sub structure has been developed. As mentioned above, a Firefox web-browser plugin has been designed to access publications in the PSIRP overlay. This plugin provides the functionality necessary to retrieve publications transparently by entering a PSIRP URL into the browser's address bar in the same way as entering HTTP URLs. Once the client sends a PSIRP URL to the network, the client subscribes to the content based upon a unique RID and SID pair. The rendezvous system receives the request and returns the metadata for this publication. The client subscribes for the publication based on this metadata. Ultimately, the publisher delivers the resulting data to the Firefox web browser. For example, if the data is video, a VLC media player can play the received content.

6.6 CouchBase

Here we give some details about CouchBase, which is a NoSQL document database we used in this project. In order to implement the cache functionality of our proxy, we utilize the Couchbase Server 2.0. The Couchbase Server supports JavaScript Object Notation, indexing and querying, incremental Map, and cross datacenter replication. We deployed the Couchbase server on a Linux platform. Compared to other databases, CouchBase has some advantages such as a flexible data model, easy scalability, consistent high performance and "Always online" features. The end users could benefit from these features when they are using applications with Couchbase. For instance, the latency can be lower and the throughput may be higher. In our system, we decided to do ten experiments and calculate the average of these experiments as the final result for each data point. Initially, ten buckets are built as the cache of the experiment. Each experiment uses one empty cache. We flushed each buckets after the experiment so that the initial status of the cache is empty.

6.7 Content Transmission

In the following sub-sections, we present the content forms that the proxy and pub/sub overlay supports. Each proxy has a local cache to store information items. Once a fresh information item is received by the proxy, the proxy will determine whether the content is cacheable depending on freshness, validation, and invalidation. If the proxy caches the content, then it not only gives the content a key with an HTTP format, but it also assigns a PSIRP key as well. Hence, it is possible to use both PSIRP and HTTP URLs to search for this same content. Currently, our proxy supports file transfer, video streaming, and voice communication.

6.7.1 File Transfer

File exchange is one of the most important forms of Internet communications. The basic interaction between a publisher and subscriber in Blackadder is a push model. Specifically, the core components of Blackadder push data from a publisher to subscriber along a forwarding path. Blackadder has a framework for multipath receiver-driven transport, which is utilized for file transfer. This framework is pull-based. In this approach, the subscriber requests a file piece by piece. The receiver publishes pieces in response to requests from the subscriber. Moreover, the requested pieces can be transmitted from a publisher (or publishers) to subscriber via multiple paths.

In our scenario, every node in the pub/sub overlay publishes the scope and announces a content item’s availability to the rest of system. Clients simply subscribe to the scope or the specific information item if they are interested. Based upon the identifiers, the system matches the publication and subscriptions and generates a FID to forward the data from source to destination. A MULTIPATH strategy has been developed. When the rendezvous node receives subscriptions, it requires the topology manager to compute more than one available path and their associated FIDs between the publisher and subscriber. This mechanism brings some benefit such as increasing cache hit ratios and optimization of bandwidth usage.

The MULTIPATH content fetching process is shown in Figure 6-5. Initially, there are three content resources in the network. Step 1, publishers publish the availability of content to the rendezvous system. Step 2, the client is interested in this content so it subscribes to this information item via the rendezvous node with a MULTIPATH strategy. Step 3, the rendezvous node matches the publication and the subscription, and then asks the topology management system to computes multiple paths between publishers and clients. Step 4, the topology manager sends notifications to the publishers which include FIDS for each path. Finally, the publishers start publishing data to the subscriber.

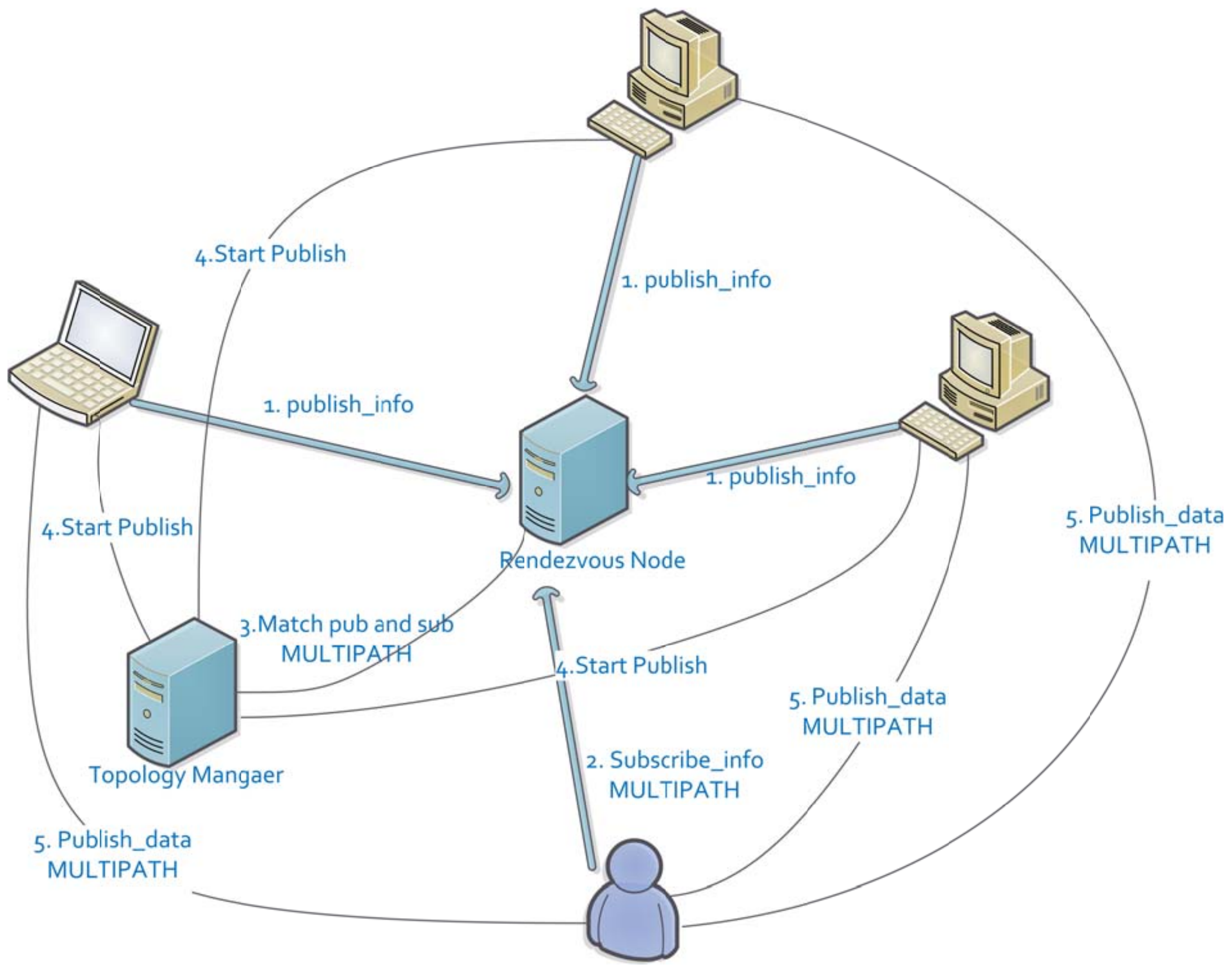

4.Start Publish
1. publish_info
1. publish_info
1. publish_info
4.Start Publish
Rendezvous Node
5. Publish_data MULTIPATH
3.Match pub and sub MULTIPATH
4.Start Publish
5. Publish_data MULTIPATH
Topology Mangaer
2. Subscribe_info MULTIPATH
5. Publish_data MULTIPATH

Figure 6-5: The example of file transfer with MULTIPATH strategy (Adapted from figure 18 of [22]).

6.7.2 Video and Voice

We developed the partly function of publishing and subscribing to video and media streams in the system. However, we have not evaluated and tested them. Here, we will introduce some basic implementation information of them. Firstly, the video provider advertises an available video with an Information ID. Once a psirp:// URL is sent to the proxy from client, this video request is transformed into a subscription and forwarded to the Publish/Subscribe overlay. When subscribers for this specific information ID join the pub/sub system, the Rendezvous system requires the content owner to start publishing data. The publisher could stop publishing the video if every subscriber unsubscribes from the scope which the video belongs to. Subscribers receive the required video frames and forward them to a VLC client to play.

We can also publish Internet media streams, such as radio and YouTube media. Before running Blackadder to transmit such streams, we need to install VLC, build a Java binding, and download the necessary jar files. Put the commons-codec-1.7.jar, vlcj-1.1.1.5.1 jar, jna.jar and platform.jar to the paths *blackadder-master/lib/* and *blackadder-master/java-binding/lib/* and *PROJECT_PATH/lib*.

A voice telephony application has been developed in Blackadder [24]. We plan to merge this application code into the proxy’s functionalities in order to enable a voice conference function. Clients can make a call over a Publish-Subscribe Internetwork (VoPSI), as it supports connection establishment between two communicating hosts and the voice packet transmission is bidirectional.

7 Evaluation

The developers of PURSUIT project have done a lot of qualitative and quantitative analysis of their implementation of their pub/sub architecture. They evaluated the security and privacy issues of their implementation of the Blackadder prototype. Moreover, the memory management performance, topology management system, and network attachment performance were evaluated in their deliverable D4.2. More details can be found in [25]. In our project, we measure the system's usability and performance from the client's point of view. We also analyze the network's performance from the proxy's view. The specific parameters that we will use in the evaluation are described below.

7.1 General Performance Analysis

Due to the combination of HTTP request and PSIRP request processing of our proxy, the performance may differ depending upon the type of the request. We will test the system with both types of requests, sending only HTTP requests, and randomly sending HTTP and PSIRP requests.

After the proxy receives a HTTP request from a client, it will search its local cache. In our analysis a hit means that the information requested by a client is already available in the proxy's cache. In some circumstances, the original server may need to be contacted to verify the freshness of the content. Over a certain period (to be stated), we will calculate the Document Hit Ratio (DHR) that is the number of hits divided by the total number of requests received by the proxy. We can also compute the Byte Hit Ratio (BHR) which indicates the percentage of the bytes delivered following hits divided by the total size of all replies sent by a proxy within a certain time interval. For our measurements we set the timer period to different number based on the number of requests.

A real network usually has more than one client sending requests to a proxy at the same time. Hence, we want to assess the ability of our proxy to process multiple concurrent requests as part of our evaluation of the proxy's performance. The proxy needs time to respond and process each individual request. Generally, the more concurrent requests a proxy can handle within a given time interval, the more powerful it is. In reality, the ability to deal with concurrent requests is based on multiple factors, such as traffic load, memory capacity, and disk capacity of the proxy.

7.1.1 Parameters of the proxy's performance

In this subsection we consider a number of different parameters of the proxy's performance. This analysis looks at the caching behavior and the characteristics of the proxy's traffic.

7.1.1.1 Cache Analysis

The Cache Hit Ratio (%) is the most important metric for measuring the effectiveness and demonstrating the value of a proxy server. The cache hit ratio represents the percentage of client requests that the proxy server's local cache satisfied. The cache hit ratio indicates how many requests are hits. Both the Document Hit Ratio (DHR) and Byte Hit Ratio (BHR) are possible to calculate in our network. However, the measurements of effects on bandwidth and latency cannot be seen from DHR. A higher DHR reflects a shorter request response time (RRT) while a higher BHR means lower network bandwidth cost and more efficient disk bandwidth usage. Caching large objects makes a greater contribution to the BHR than caching small objects. Some improvements can be made by using better cache management, such as changing the caching policy.

To estimate the network load and bandwidth requirements, we can consider the transfer sizes of the objects as a parameter. Earlier research shows that large documents are less popular than small sized documents. More details can be found in [26]. About 99% of transfers are smaller than 64 KB. In a real scenario, the client may send different types of data requests, such as requesting video, files, real-time chatting, and voice. Some types of data are sensitive to variance in transit time. Video and audio data are particularly sensitive. The type of cached content influences the disk capacity required by the proxy to achieve a given DHR or BHR. The performance of the proxy is also affected by the total amount of traffic; hence we should evaluate the proxy under different traffic load conditions.

7.1.1.2 Proxy's Traffic

The proxy acts as intermediary to pass a request from a client to a remote server. The client can send both HTTP and PSIRP requests to the proxy, so it is easy to count the number of HTTP and PSIRP requests. If a cache with a suitable size has been setup in the proxy, the response time should decrease with increasing hit rate. Hence, we measure the Request Response Time (RRT) as a function of the hit rate.

Request Response Time	We calculate the Average RRT for both scenarios (HTTP and PSIRP requests). RRT is measured from the beginning of a send function call to the time the client receives the response from local proxy. This metric can be a good indicator of queuing and congestion. A change in RRT is usually an indication of a change of configuration or congestion level.
Server request count	If the content does not exist in the local cache, then the proxy will forward the request to a remote server or subscribe to it in the PSIRP overlay. The Server Request count is the number of requests that the proxy sends to remote servers, including PSIRP overlay neighbor caches. We count the total number of HTTP and PSIRP requests individually. The server request count can be easily calculated by using total number of requests multiplied by the DHR.

7.1.2 Hardware and software configuration used for testing

The computer used for testing was a DELL Latitude E5520 laptop equipped with a model Intel® Core™ i5-2540M CPU @ 2.60GHz * 4 and 8 Gbytes of memory. The computer was running the Ubuntu operating systems version 12.10. The Java version we used for testing is 1.6.0_37 and the version of Couchbase server is 2.0. We use the Blackadder prototype v0.3 which is released by PURSUIT project on November 2012.

7.1.3 General performance of random sized files

When clients want to obtain files from the Internet, they will send HTTP requests to a remote server. In this scenario, we use the IETF's Request for Comments (RFC) Index website for testing. The reason that we choose this website is because the RFCs consist of more than 6 800 text (TXT) files with different sizes. Moreover, the prefixes of the RFC URLs are same, so if client wants to fetch a specific file and he/she knows the number of the RFC, then it is only necessary to append the RFC number to the prefix URL. In our evaluation, the system randomly chooses a set of optional URLs (100,200,300,400,500) in advance. After that, the client randomly sends HTTP requests from the collection of pre-choosen URLs within a certain period of time (specifically the three intervals: 30s, 60s, and 90s). Once a request arrives at local proxy, the system will process the request and wait for the reply. Then the response data would return to client. In the meanwhile, the client keeps waiting for the reply and does not send other request. Finally, the requested information is received by client. This is a whole content request process. In our evaluation, the client repeats this process constantly within the testing duration. The request rate is a certain number. For example, if the client sends 3KB HTTP files within 30s, the request rate is 0.87 requests/second. The cache will be flushed after every experiment. Every data from the table does not share any connections or testing time with each other.

For this pub/sub proxy for HTTP, we setup a context to test the hit rate of our cache. The cache stores the files and then publishes them to the PSIRP overlay. After the PSIRP content exists in the overlay, the proxy returns the catalog of existing contents to the client. The client could request both HTTP URLs and the PSIRP URLs. We set the refresh interval to 1s. The choice of the appropriate interval is based on the processing time and the traffic load. Based on the pre-test, the frequency of cache update is 1 item per second on average. So we choose 1 second as refresh interval to make sure the client could get the latest cache view. We compare DHR and BHR when the client sends only HTTP requests and HTTP plus PSIRP requests.

7.1.3.2 Random HTTP Request

Table 7-1 and Table 7-2 (plotted in Figure 7-1and Figure 7-2) show the DHR and BHR results of the client randomly choosing URLs from 100 URLs, 200 URLs, 300 URLs, 400 URLs, and 500 URLs each experiment. The duration of tests were 30s, 60s and 90s. For a given number of optional HTTP URLs, we can see that the Average DHR and BHR show an increasing trend as the testing time increases from 30s to 90s. The Couchbase server saves all the previously requested content in the cache. A longer test duration leads to more contents are likely to already be in the cache. When the proxy receives a HTTP request, it always searches the local cache firstly. So the DHR and BHR are higher along as the testing time increases. If the testing time is same, the Average DHR and BHR show a decreasing trend as the total optional number of URLs increases from 100 to 500. The cache strategy could be changed based on the requirements of clients and size of storage capacity.

Table 7-1: Average DHR of random sized HTTP requests

Average DHR	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	0.2833	0.1097	0.0678	0.0594	0.03
60 s	0.2872	0.176	0.1049	0.1058	0.0643
90 s	0.554	0.2601	0.1538	0.115	0.0949

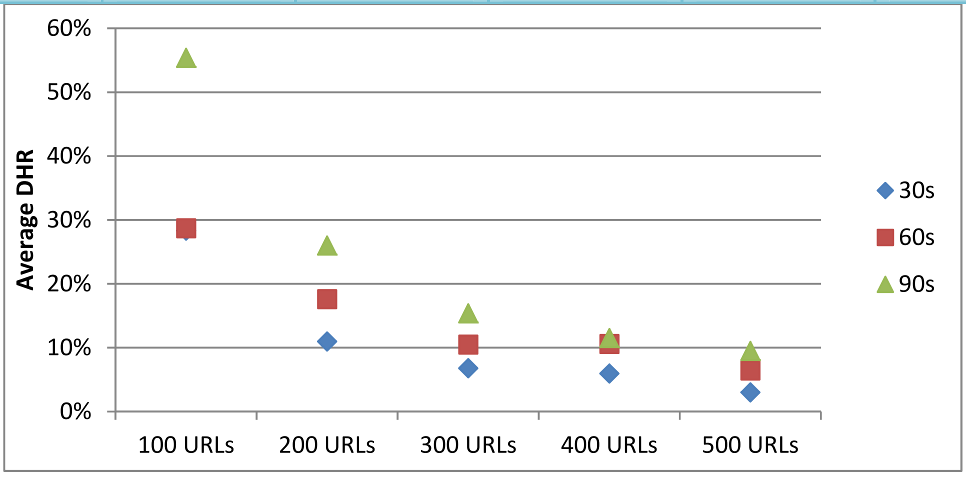


Figure 7-1: Average DHR of random sized HTTP requests

Table 7-2: Average BHR of random sized HTTP requests

Average BHR	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	0.2805	0.1338	0.0404	0.0524	0.0157
60 s	0.279	0.1663	0.0954	0.1219	0.0592
90 s	0.5643	0.2769	0.1516	0.1289	0.0923

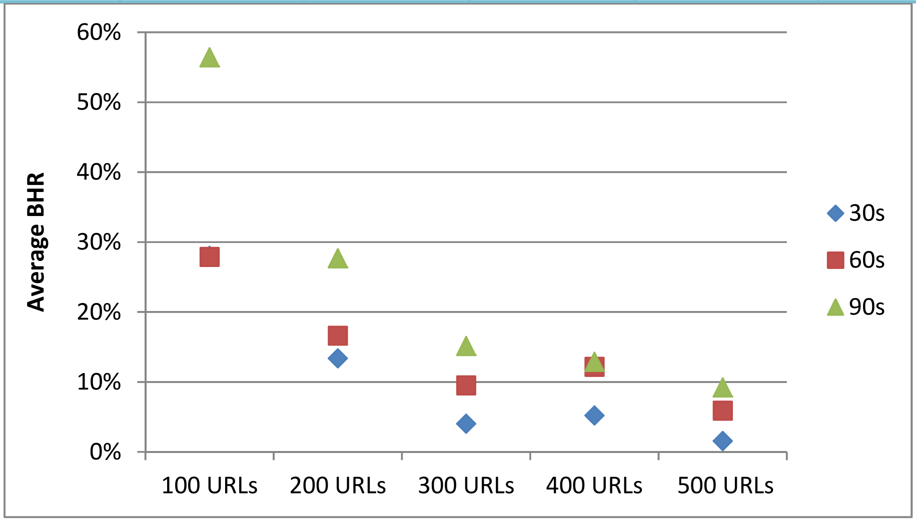


Figure 7-2: Average BHR of random sized HTTP requests

Table 7-3 depicts how the average RRT is influenced by the hit ratio. For a given testing time and number of URLs, a higher hit ratio means more content can be retrieved from the local cache. This eliminates the time required to request, retrieve, and process requests that need to be sent between the proxy and remote server. The higher the hit ratio is, the shorter the RRT. For instance, compared to the result of (300 URLs in 90s), the result of (200 URLs in 90s) has a higher average DHR and a shorter average RRT. As can be seen from Figure 7-4, if the Average RRT increases, the total number of HTTP requests will be reduced.

Table 7-3: Average RRT and Total number of random sized HTTP requests

Average RRT (ms) / Total HTTP Requests	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	415/651	508/529	519/522	715/378	934/291
60 s	737/736	766/707	809/668	882/615	982/551
90 s	444/1823	722/1125	830/979	925/878	896/908

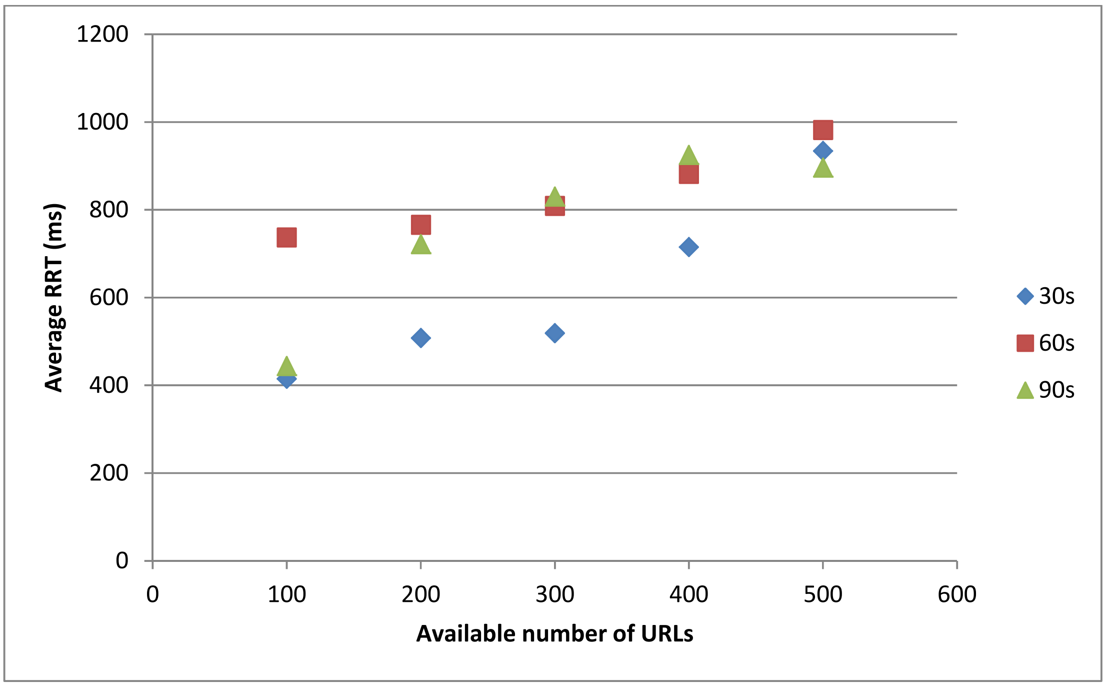


Figure 7-3: Average RRT of random sized HTTP requests

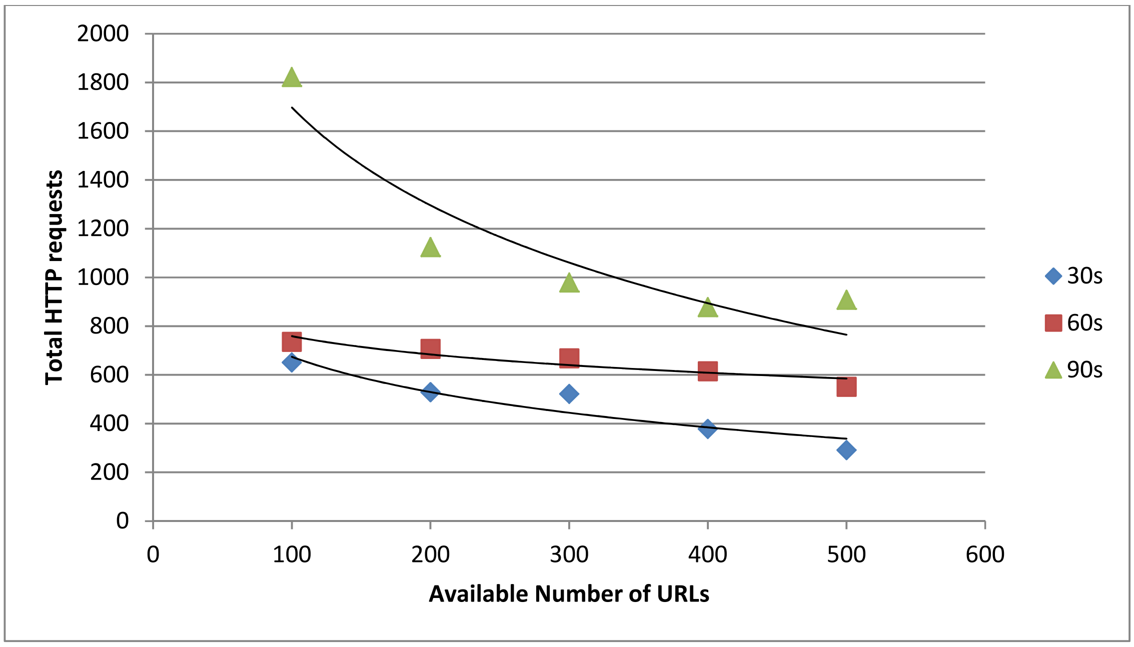


Figure 7-4: The total number of random sized HTTP requests

7.1.3.3 Random HTTP +PSIRP Request

In the former scenario, once the proxy fetches the requested content from the remote server, it automatically saves the content in its local cache and also publishes the content to the PSIRP overlay. However, in the previous scenario the clients only send HTTP requests. In this paragraph, we enable the PSIRP processing function of proxy so that the client can randomly send both HTTP and PSIRP URLs. Initially, the list of PSIRP URLs is empty. When the proxy receives a request for HTTP content, it publish the metadata of the content to the PSIRP overlay and adds a fresh PSIRP item to the available link catalog. The client requests and refreshes the PSIRP catalog once every second. Every proxy in the PSIRP overlay could subscribe to a catalog from other proxies. The proxy returns the requested catalog to the clients. The clients randomly choose either a HTTP or PSIRP request from the catalog.

Table 7-4 and Table 7-5 (plotted in Figure 7-5and Figure 7-6) show the average DHR and BHR when the clients randomly choose HTTP and PSIRP URLs. The average DHR and BHR share the same trend as the results of the scenario when the clients only send HTTP requests. When the client requests a PSIRP URL, the content already exists in the local cache or PSIRP overlay, hence we did not include the PSIRP content hit ratio in these results (as it would always be 100%). We only utilize one proxy in the testing process due to hardware limitations (in terms of the number of virtual machines that we could run on the computer used for testing). However, more proxies could be added to the PSIRP overlay, in this case the hit ratio of PSIRP content would be another metric.

Table 7-4: Average DHR of random sized HTTP and PSIRP requests

Average DHR	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	0.2278	0.1286	0.0683	0.039	0.035
60 s	0.343	0.1544	0.1054	0.0615	0.0521
90 s	0.5809	0.2367	0.1504	0.0995	0.0832

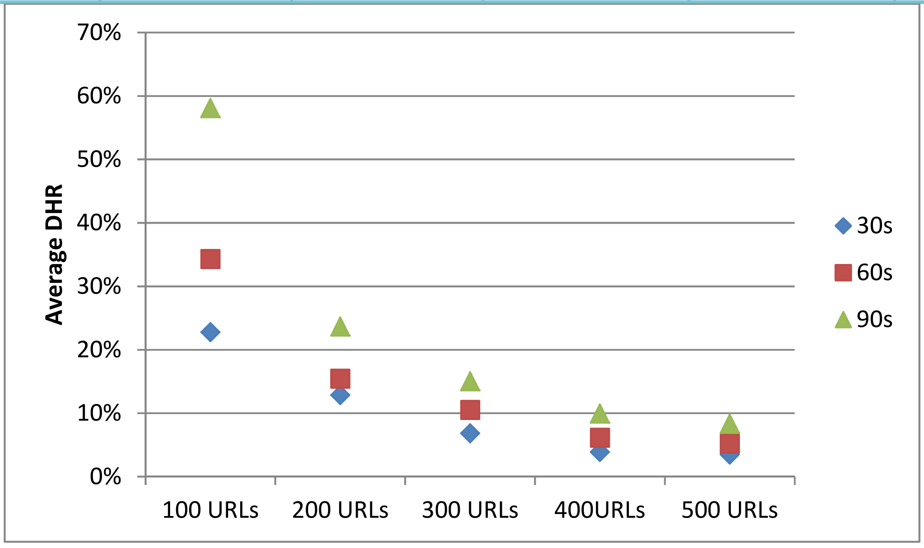


Figure 7-5: Average DHR of random sized HTTP and PSIRP requests

Table 7-5: Average BHR of random sized HTTP and PSIRP requests

Average BHR	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	0.2377	0.1277	0.0649	0.0318	0.0234
60 s	0.36	0.1666	0.0952	0.0645	0.0621
90 s	0.5798	0.2186	0.1416	0.0904	0.101

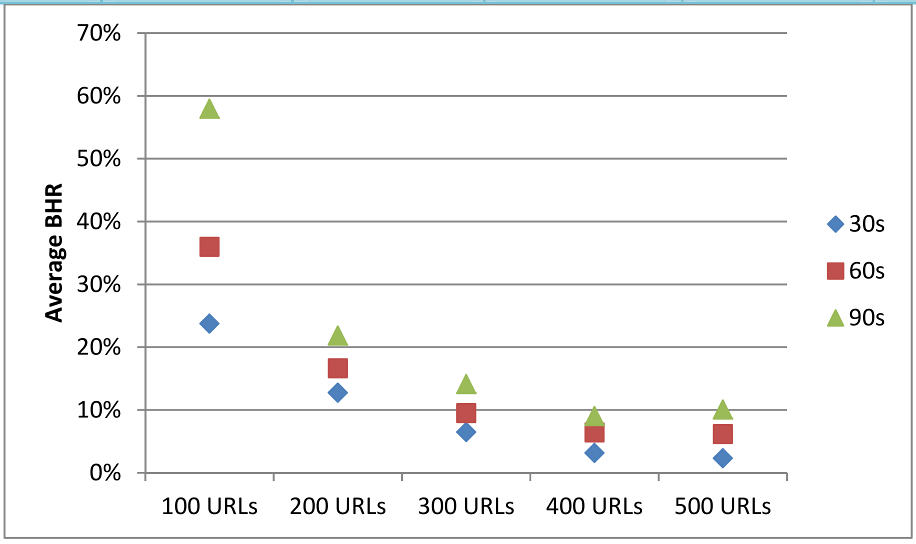


Figure 7-6: Average BHR of random sized HTTP and PSIRP requests

Table 7-6 shows the results of testing in terms of the total number of random size HTTP and PSIRP requests under different situations. It can be seen from these three figures (Figure 7-7, Figure 7-8, and Figure 7-9), the percentage of PSIRP requests of the total number of requests increases as the testing time becomes longer. Consider 100URLs as an example, depending on the testing duration, the proportion of PSIRP requests increases from 26% to 47%.

Table 7-6: The total number of random sized HTTP and PSIRP requests

PSIRP requests/HTTP requests	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30s	190/553	135/551	107/533	35/367	18/305
60s	457/828	201/671	105/674	82/560	40/567
90s	1715/1912	451/1085	281/955	156/835	160/904

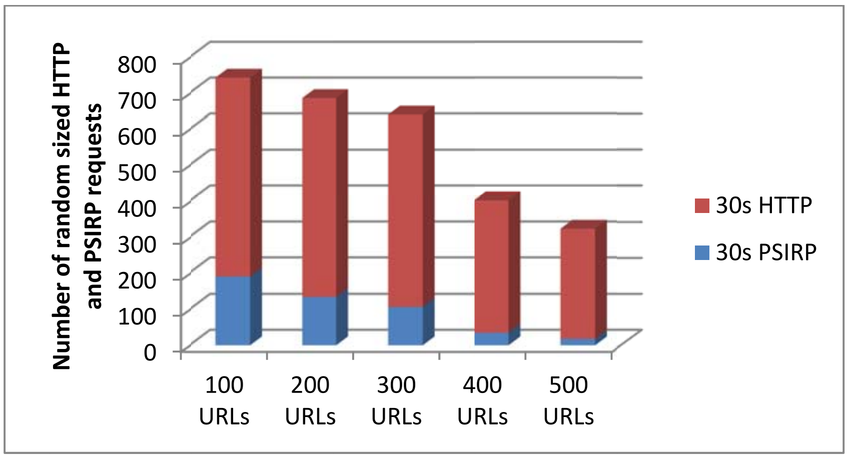


Figure 7-7: Total number of random sized HTTP and PSIRP requests (30s)

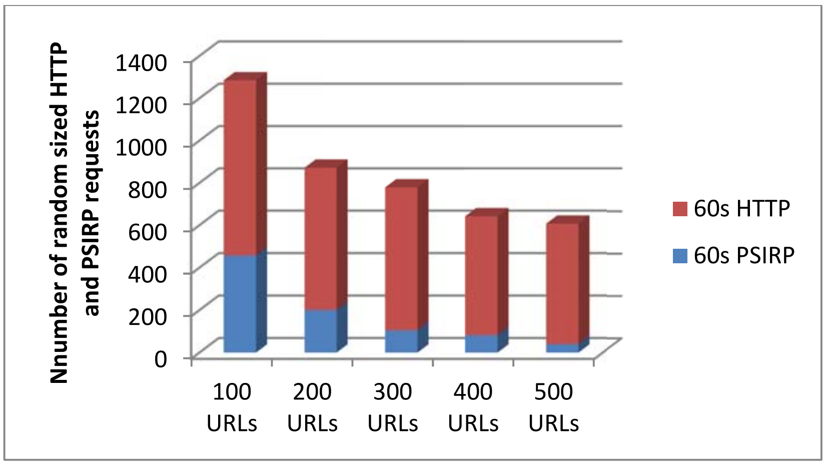


Figure 7-8: Total number of random sized HTTP and PSIRP requests (60s)

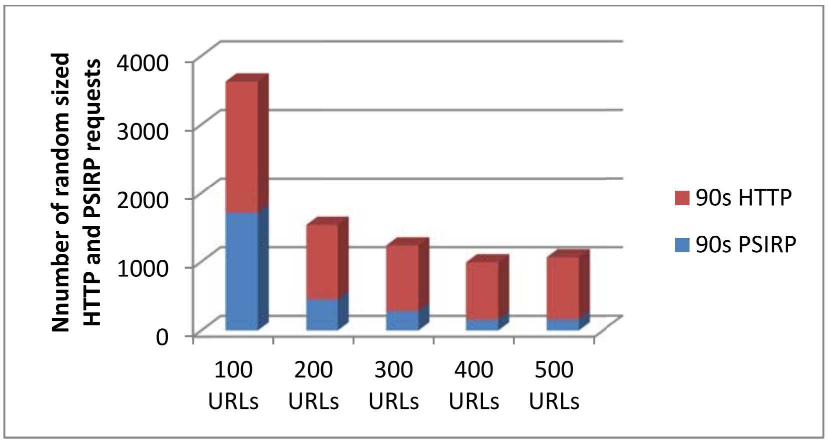


Figure 7-9: Total number of random sized HTTP and PSIRP requests (90s)

Table 7-7 depicts the Average RRT when the client sends both HTTP requests and PSIRP requests. For the same test duration, the average RRT grows as the total number of available links becomes bigger. This situation occurs simply because of the fact that more and more contents need to be fetched from remote server. A higher hit ratio means a shorter RRT.

Table 7-7: Average RRT of random size HTTP and PSIRP requests

Average RRT (ms)	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30 s	489	489	506	737	890
60 s	653	806	800	1051	959
90 s	423	746	850	972	898

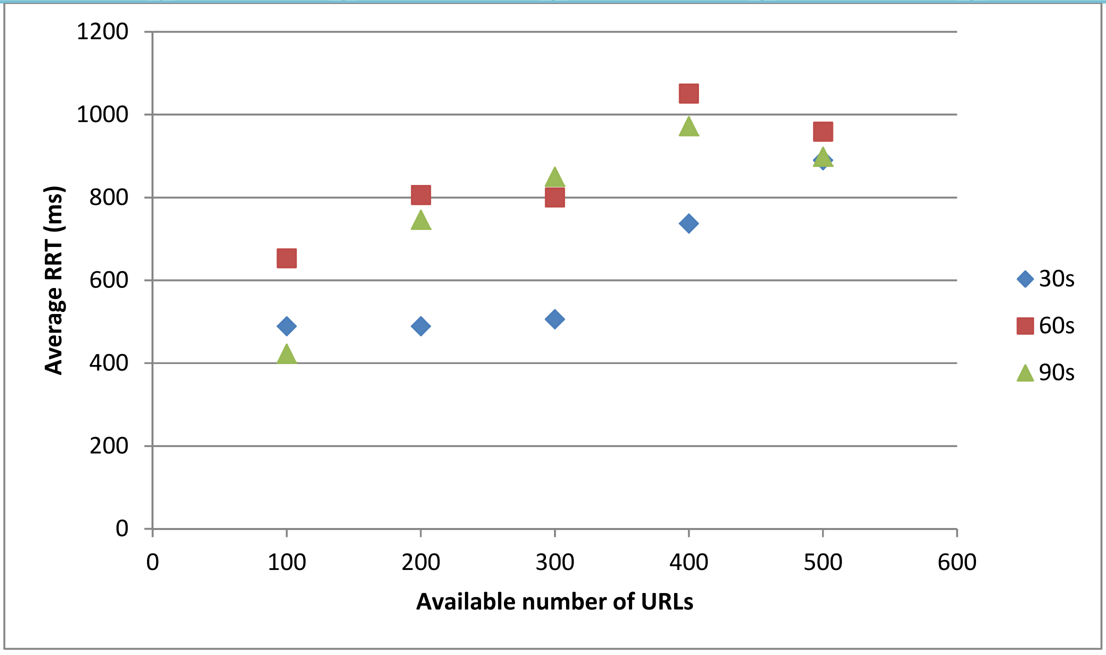


Figure 7-10: Average RRT of random sized HTTP and PSIRP requests

7.1.4 General performance for fixed sized files

This subsection will evaluate how the transfer sizes of object influence the performance of the system. To consider and estimate the network load and bandwidth requirements, we utilized different sizes of objects as the file resource and each type of files has 200 different URLs. The file size ranged from 3KB to 2.3MB. The test durations were 30s, 120s, and 240s. Since the size of files is fixed, the DHR and BHR result should be same. As in the previous section we have two groups of experiments: only HTTP requests and HTTP plus PSIRP request. For the former one, once the proxy receives the HTTP content from remote server, it publishes the content to the PSIRP overlay. However, the clients only ask for HTTP contents. This group of experiments will be used as comparison for the second group.

Usually, there is a limited size for cached files. This limitation prevents users on slow connections from storing large images and videos in the cache. The appropriate cache strategy is based on the requirements of the clients. For our evaluation, we use 3KB, 55KB, 188KB, 608KB, and 2.3MB files to represent the "normal size" of text files.

7.1.4.1 Fixed sized HTTP Requests

The results shown in Table 7-8 reflect several trends. For a given size of file, when the testing time increases from 30s to 240s, the general trends of Average DHR or BHR appear to increase. For instance, we can see from the first numbers of all three lines that the Average DHR has increased remarkably from 7.23% in 30s to 44.26% in 240s. A longer testing period means more contents are stored in the local cache. So we expect that the DHR is higher for a longer testing period. A very noticeable trend is the steady decrease in each set of data as the test duration increases. The graph proves that smaller size files have a higher document hit ratio. Because for a given period of time with a stable Internet transfer rate, the throughput of the network is also stable. No matter how big the file is, the network can only handle traffic up to the limit of the available bandwidth (thoughput). Consider the series of tests, each with a duration of 120s; the Average DHR for 3KB files reached nearly 26%, while the Average DHR of 2.3MB file declines to only 7%.

Table 7-8: Average DHR/BHR of fixed sized HTTP request

Average DHR/BHR	3KB	55KB	188KB	608KB	2.3MB
30s	0.0723	0.05618	0.02112	0.019	0
120s	0.2583	0.1755	0.1369	0.1096	0.07
240s	0.4426	0.4442	0.3404	0.2095	0.1331

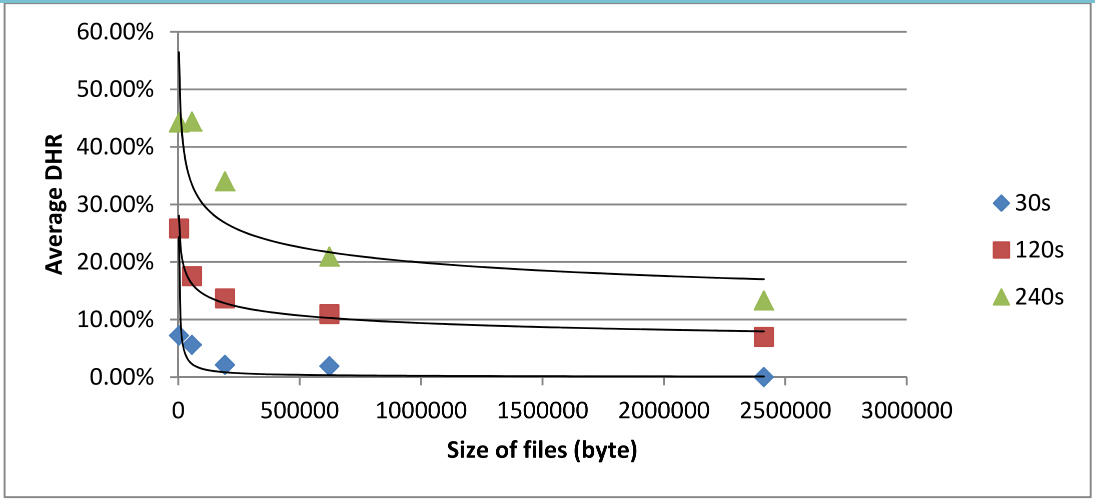


Figure 7-11: Average DHR/BHR of fixed sized HTTP request

In the Figure 7-12, we can see that the Average RRT swells as we increase from 3KB, 56KB, 188KB, and 608KB, peaking at 2.3MB condition. Obviously, the growth trend happens in each set of experiments (30s, 120s, and 240s). The reason for this trend is larger files take more time to transfer both from remote severs to the proxy and from the proxy to a client. Hence, within a limited period, clients could request fewer 2.3MB files compared to smaller size files. As can be seen in Figure 7-13 the number of HTTP requests decreases as the size of files increases.

Table 7-9: Average RRT and Total quantity of fixed sized HTTP requests

Average RRT (ms) / Total HTTP requests	3KB	55KB	188KB	608KB	2.3MB
30s	1155/261	1529/188	1959/143	2631/106	4324/66
120s	944/1341	1309/924	1879/578	2491/439	4230/258
240s	954/2266	894/2421	1316/1646	2600/841	3705/587

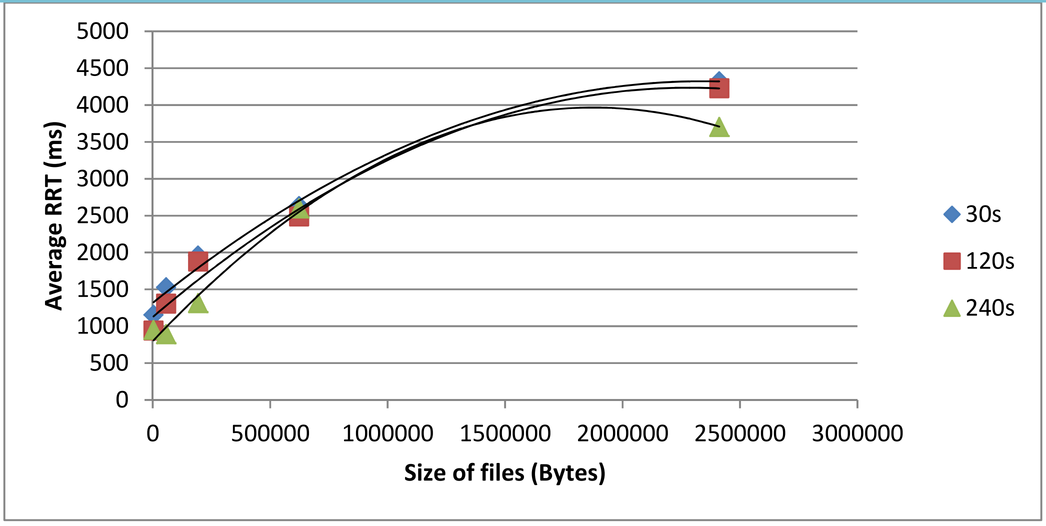


Figure 7-12: Average RRT of fixed sized HTTP requests

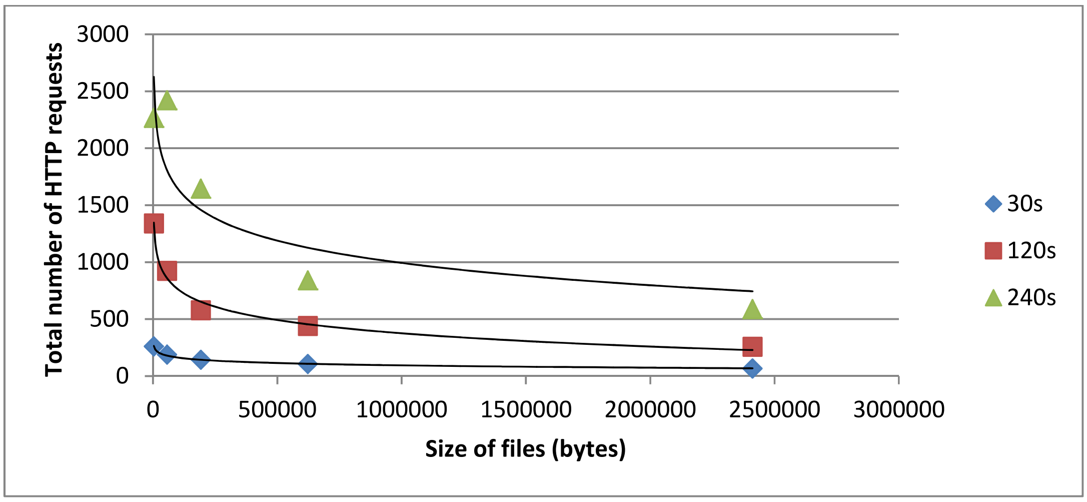


Figure 7-13: The total quantity of HTTP requests (Fixed size)

7.1.4.2 Fixed size of HTTP and PSIRP Request

This section represents the test result when clients randomly send HTTP requests and PSIRP requests for fixed size text files. The DHR, BHR, total quantity of HTTP and PSIRP requests, and the RRT are evaluated in this section. The context is similar to the requests for random sized files. The only difference is the file resources are set to fixed size in the range from 3KB to 2.3MB.

Comparing Table 7-10 (and Figure 7-14) to Table 7-8 (Figure 7-11), the Average DHR and BHR share same trend with the result of the situation when the clients only send HTTP requests. When the testing period is set to 30s, the DHR of 3KB files has reached to 67.1% and the number drops to 14.7% when the size of files increases to 2.3 MB. However, it is almost impossible to reach such a high level of DHR for a real cache. We consider how the size of file influences the DHR.

Table 7-10: Average DHR/BHR of fixed size HTTP and PSIRP requests

Average DHR/BHR	3KB	55KB	188KB	608KB	2.3MB
30s	0.0657	0.0591	0.0288	0.0283	0
120s	0.248	0.1811	0.147	0.1121	0.0638
240s	0.6706	0.4494	0.3246	0.2498	0.1466

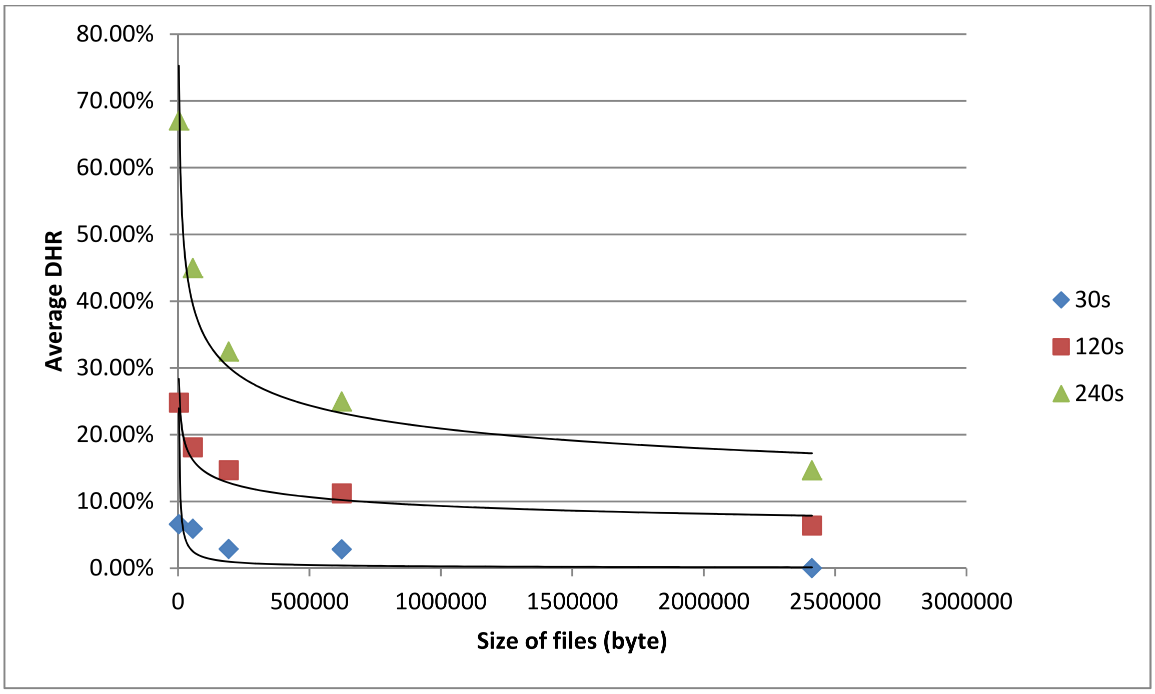


Figure 7-14: Average DHR/BHR of fixed size HTTP and PSIRP requests

Table 7-11 shows the Average RRT and Table 7-12 shows the quantity of HTTP plus PSIRP requests in the different intervals of time. As Table 7-11 shows the RRT goes up as the file size increases. Larger files require more time to transfer from a remote server to a client or from a publisher to a subscriber. Within a limited time period, clients could request fewer 2.3MB files as compared to a smaller size file scenario. Moreover, if we change the testing duration, but consider the same size of resource, the Average RRT drops to a markedly low level. For example, the 3KB Average RRT is 1171ms (in 30s) and it declines to 426ms (in 240s), this means that more contents are stored in the local cache or can be subscribed to from the PSIRP overlay.

Table 7-11: Average RRT of fixed HTTP plus PSIRP requests

Average RRT (ms)	0	3KB	55KB	188KB	608KB	2.3MB
30s	1129	1171	1462	1995	2646	4333
120s	894	944	1263	1855	2428	3886
240s	381	426	870	1452	2002	3753

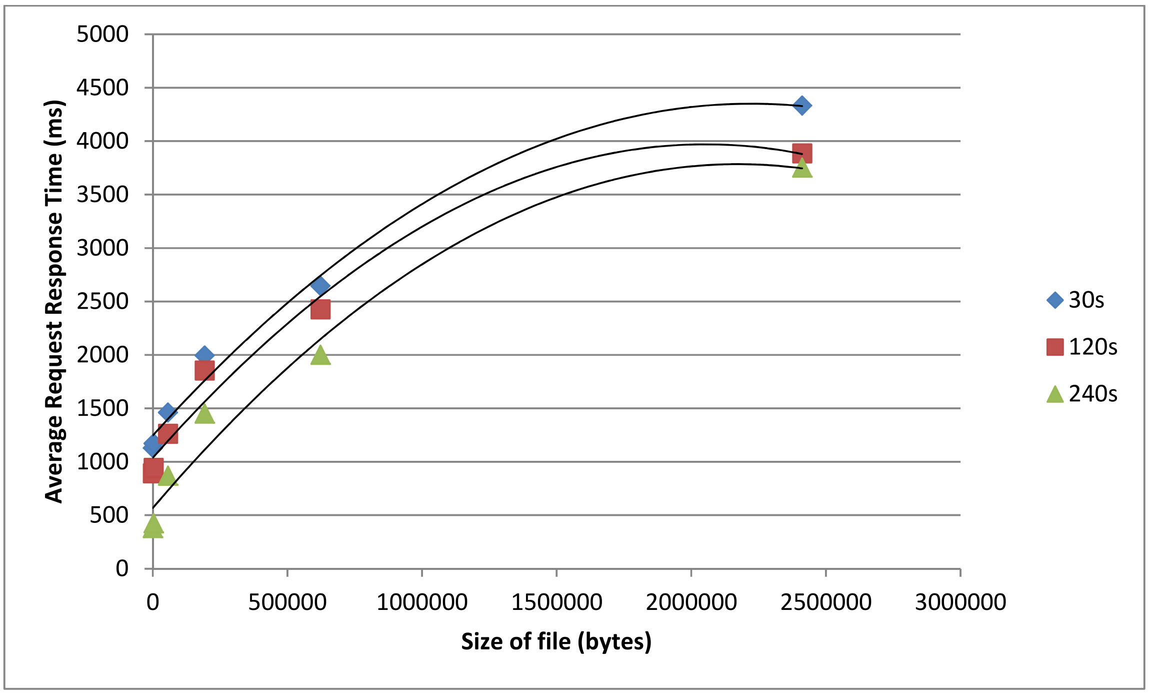


Figure 7-15: Average RRT of fixed HTTP plus PSIRP requests

As can be seen clearly in the Table 7-12, the quantity of HTTP/PSIRP requests declined markedly from 4586/5144 with 3KB files to only 132/581 with 2.3MB files (240s). A trend that should be noticed is that with the same size of text file resource, the percentage of PSIRP requests serviced in 240s of testing is much higher than during 30s of testing. This occurs because more contents are published to the PSIRP overlay as the testing duration increases. As clients refresh their copy of the catalog, the PSIRP request hit ratio rises.

Table 7-12: Total number of fixed HTTP and PSIRP requests

PSIRP requests/HTTP requests	3KB	55KB	188KB	608KB	2.3MB
30s	23/ 233	19/187	13/140	7/107	1/66
120s	498/1146	334/857	137/586	91/447	43/283
240s	4586/5144	1760/2482	804/1492	347/1082	132/581

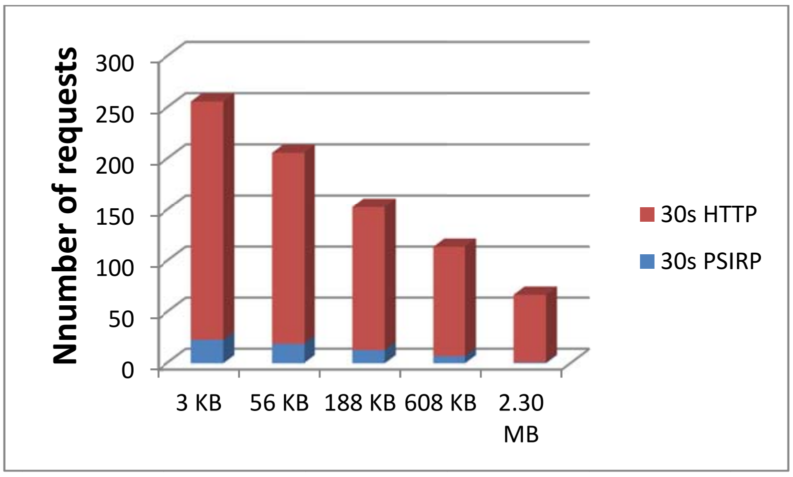


Figure 7-16: Total number of fixed HTTP and PSIRP requests in 30s

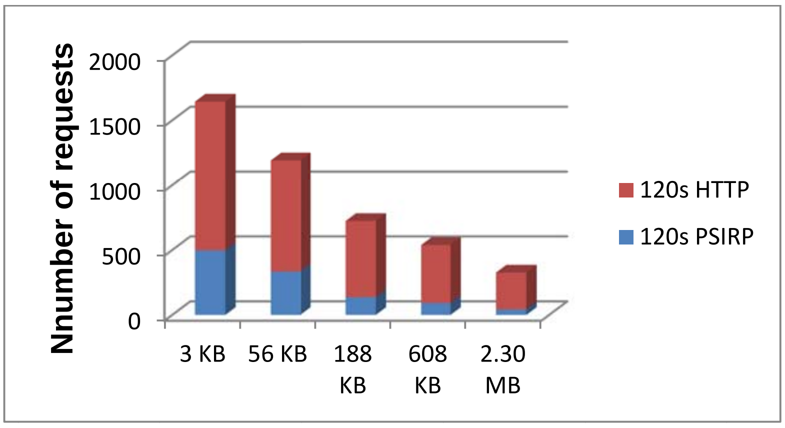


Figure 7-17: Total number of fixed HTTP and PSIRP requests in 120s

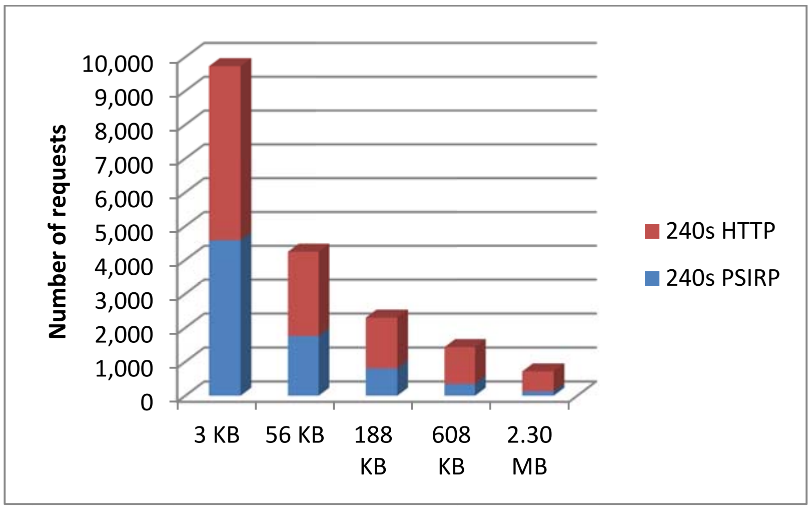


Figure 7-18: Total number of fixed HTTP and PSIRP request in 240s

7.2 Network related performance analysis

This section describes other parameters regarding the network's performance. We evaluate the proxy's performance based upon connect time and reply time for communication between the client and proxy, as well as between the proxy and a remote server.

7.2.1 Parameter Descriptions

In this section we will consider three parameters to describe the network related performance of the proxy, specifically: proxy connect time, server reply time, and server response time.

7.2.1.1 Proxy Connect Time

A client sends a new request to the proxy, and then the proxy identifies the type of URL. The Proxy Connect Time is the time that the proxy takes to forward a request to the origin server. We compare the proxy connect time for a HTTP request and PSIRP request as a function of the size of the requested content.

Average Proxy Connect Time = HTTP Total Proxy Connect Time + PSIRP Total Proxy Connect Time) / (Total number of PSIRP requests + Total number of HTTP requests)

7.2.1.2 Server Reply Time

After the proxy forwards a request to a remote server, it will wait for a reply. The duration of this waiting is the Server Reply Time. This time depends on the traffic load due to competing network traffic on the path to the server, on the load on the server (from other requests), the performance of the server, etc. The Server Reply Time will increase when network congestion increases. We measure this metric for different sizes of content and under different traffic and server loads.

Average Server Reply Time = HTTP Total Server Reply Time+ PSIRP Server Reply Time)/ / (Total number of PSIRP requests + Total number of HTTP requests)

7.2.1.3 Server Response Time

The Server Response Time is a measure of the total delay from when the remote server is sent a request by the proxy and when the proxy receives a reply. It is equal to the proxy connect time plus the server reply time.

Server Response Time = Average Server Reply Time + Average Proxy Connect Time

7.2.2 Performance of the network with random sized HTTP plus PSIRP Requests

Through the HTTP module in the system, we measured the Total HTTP Proxy Connect Time and Total HTTP Server Reply Time. The PSIRP module also has a function to record the Total PSIRP Proxy Connect Time and Total PSIRP Server Reply Time. As in the earlier tests we used the IETF RFC text files as the resources. Clients request both HTTP and PSIRP contents within a certain period (30s, 60s, and 90s).

7.2.2.1 Average Proxy Connect Time

The proxy listens to two ports, 9800 and 9801. When a request arrives at proxy, the system will process it including identifying the type of URL and searching the local cache. Once this operation is finished, it forwards the request to remote server and PSIRP overlay or directly returns the content to clients. Table 7-13 and Figure 7-19 show the average proxy connect time with different numbers of random sized files. According to the Table 7-13, this procesings takes about 1ms.

Table 7-13: The Average Proxy Connect Time for random size of HTTP and PSIRP requests:

Average Proxy Connect Time (ms)	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30s	0.9816	1.1837	0.9625	1.3209	1.3994
60s	0.9486	1.0998	0.9474	0.9050	1.1087
90s	0.6912	1.0046	1.1206	1.0555	0.8947

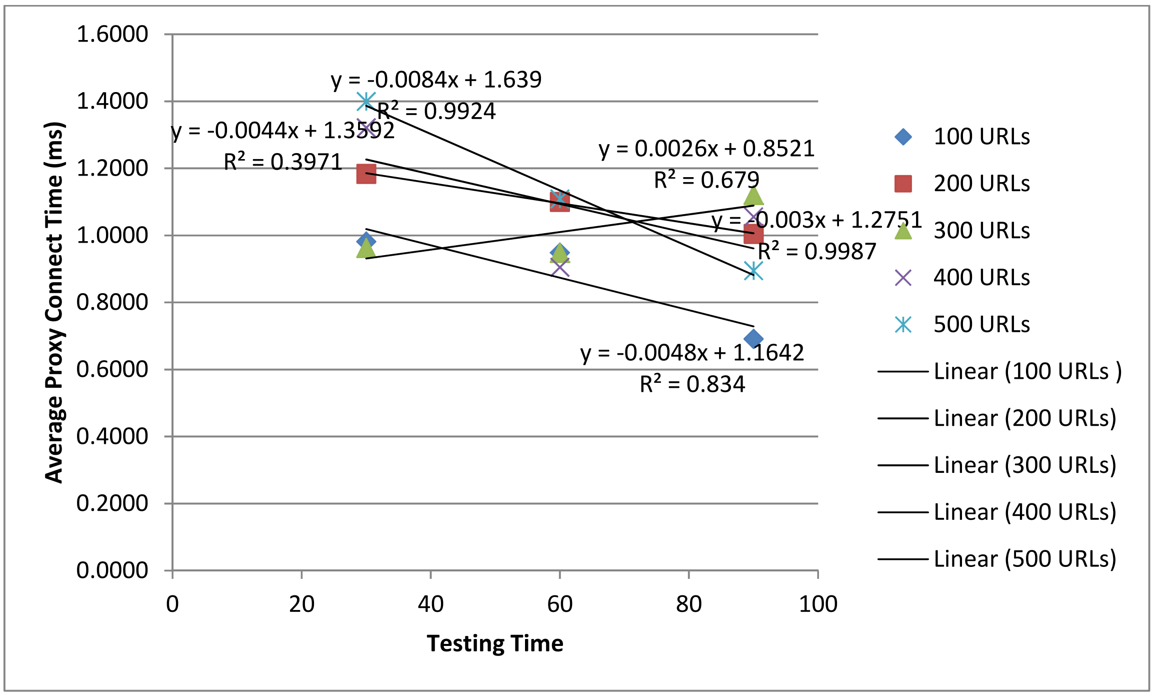


Figure 7-19: Average Proxy Connect Time for random sized HTTP and PSIRP requests

7.2.2.2 Average Server Reply Time

Table 7-14 and Figure 7-20 show the average server reply tine with different numbers of random sized files. These results indicate that the average server reply time increased dramatically when the number of text files increases. With same testing period (30s), the average server reply time for 100 optional URLs is 350ms, while the average time is 838ms with 500 URLs. A higher DHR means a lower average server reply time, because more contents can be fetched from the local cache. Since the size of files is random, the average server reply time depends on the traffic load on the path and performance of remote server or publisher proxy in the PSIRP overlay.

Table 7-14: The Average Server Reply Time for random sized HTTP and PSIRP requests

Average Server Reply Time (ms)	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30s	350.84	387.70	413.87	665.56	838.48
60s	419.72	616.26	690.41	915.86	893.22
90s	221.73	526.49	652.89	811.27	753.52

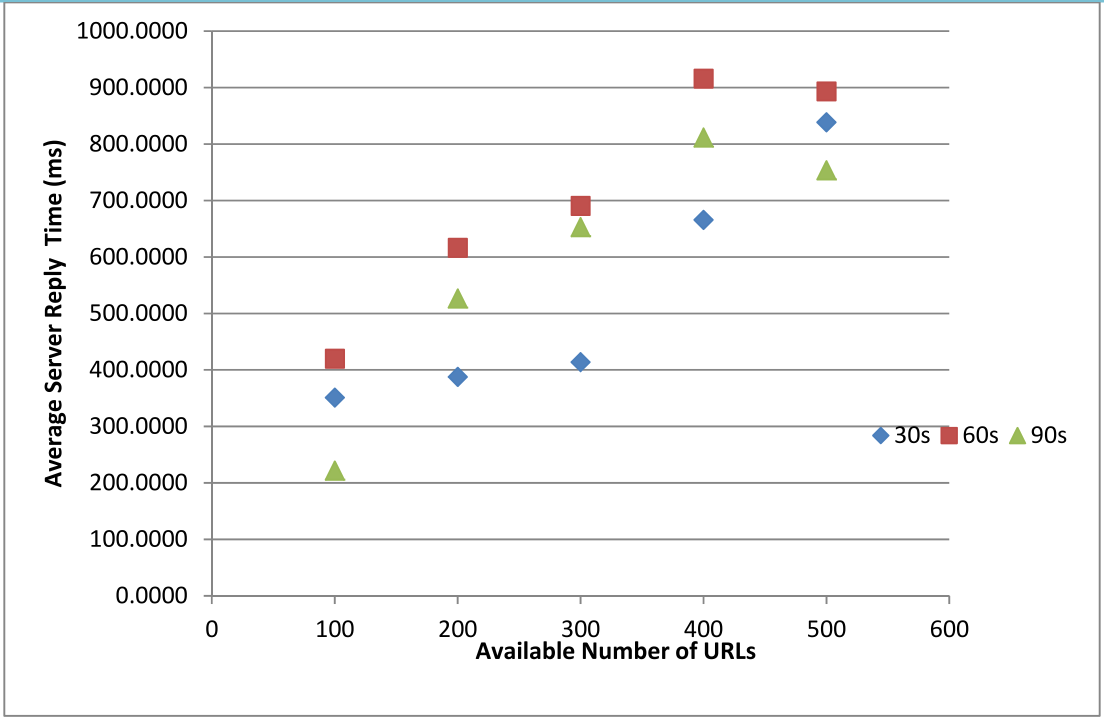


Figure 7-20: Average Server Reply Time for random sized HTTP and PSIRP requests

7.2.2.3 Server Response Time

Table 7-15 and Figure 7-22 show the average server response tine with different numbers of random size of files.

Table 7-15: The Average Server Response Time for random sized HTTP and PSIRP requests

Average Server Response Time (ms)	100 URLs	200 URLs	300 URLs	400 URLs	500 URLs
30s	351.82	388.88	414.83	666.88	839.88
60s	420.67	617.36	691.36	916.77	894.32
90s	222.42	527.50	654.01	812.33	754.41

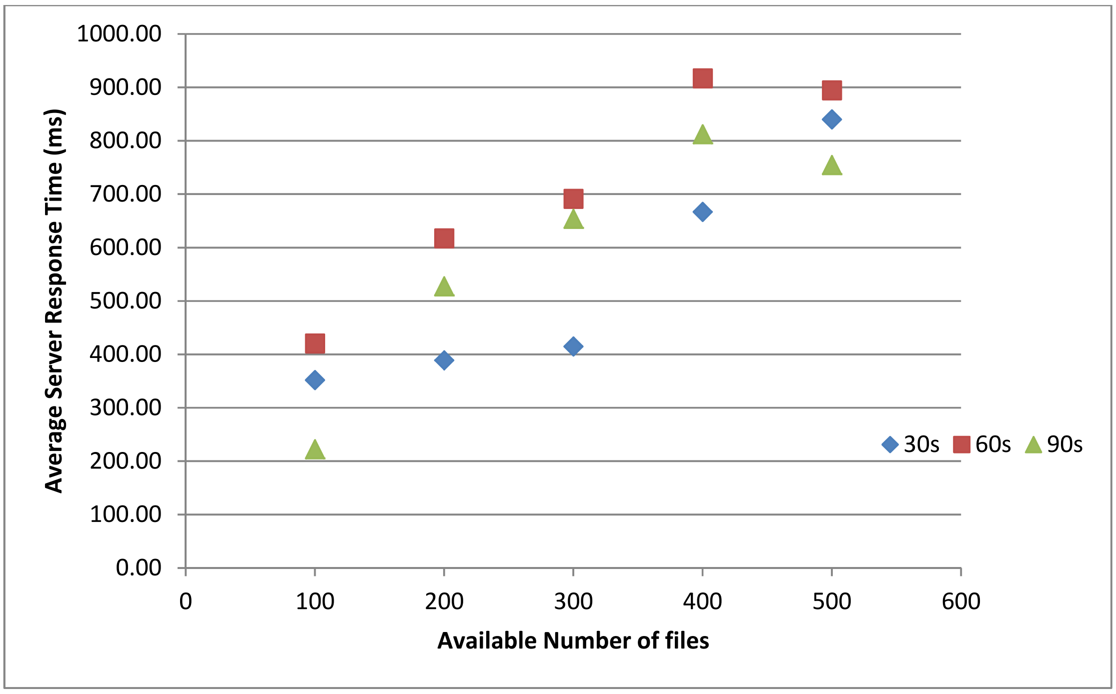


Figure 7-21: Average Server Response Time for random sized HTTP and PSIRP requests

7.2.3 Performance of the network with fixed size of HTTP plus PSIRP Requests

This section repeats the analysis of section 7.2.2, but with fixed sized file contents in the response.

7.2.3.1 *Average Proxy Connect Time*

Table 7-16 shows a clear trend of the Average proxy connect time increasing as function of the size of the requested files. As is illustrated in Figure 7-22, the larger the file's size, the longer the proxy connect time. For instance, the proxy needs 13 945 ms to process a 3KB text file (30s), while it takes nearly 50 000 ms to process a 2.3MB. The Figure 7-23 illustrates the upward trend in the average server reply time as the size of the file increases. The conclusion is similar to the RRT results we presented earlier. Both parameters share the same trend as larger files require more time to transfer, as compared to smaller files. Also, a longer testing period means a higher hit ratio, hence the average server reply time under 240s condition is shorter than the number of 30s testing period.

Table 7-16: The Average Proxy Connect Time for fixed size of HTTP and PSIRP requests

Average Proxy Connect Time (ms)	**3072**	**56320**	**192512**	**622592**	**2411725**
30s	1,3945	2,1553	2,1895	2,7807	3,9552
120s	0,9075	0,984	1,3112	1,3476	1,9969
240s	0,6369	0,7718	0,9512	1,0637	1,4418

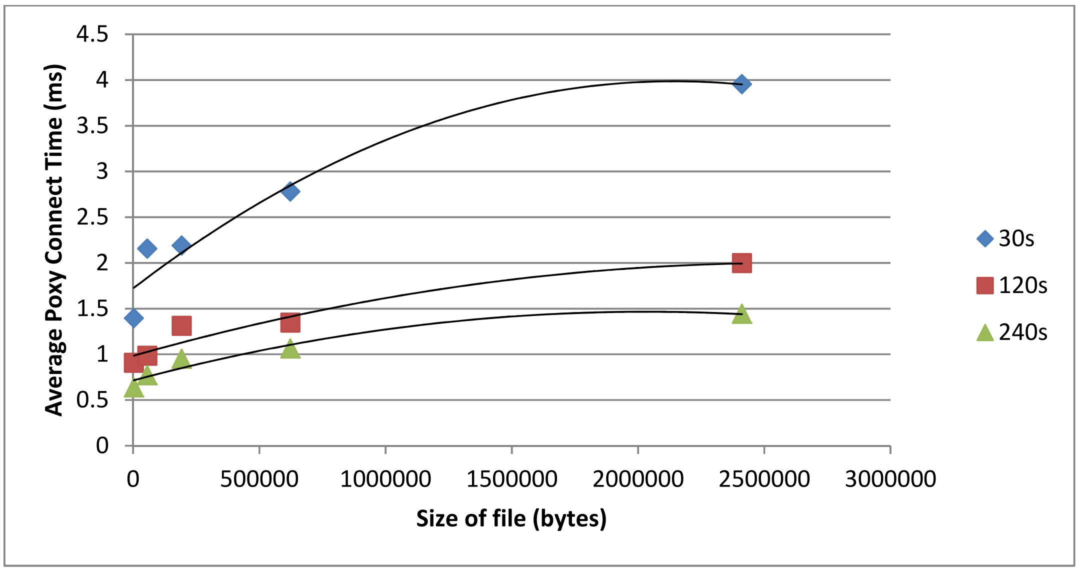


Figure 7-22: Average Proxy Connect Time for fixed size of HTTP and PSIRP requests

7.2.3.2 *Average Server Reply Time Fixed Size*

The following table and figure show the average server reply tine with different fixed size of files in several testing periods.

Table 7-17: Average Server Reply Time for fixed size of HTTP and PSIRP requests

Average Server Reply Time (ms)	**3KB**	**55KB**	**188KB**	**608KB**	**2.3MB**
30s	1057.32	1313.19	1808.88	2455.57	4290.04
120s	654.95	906.23	1505.87	2028.38	3511.98
240s	217.22	509.37	947.24	1527.87	3169.56

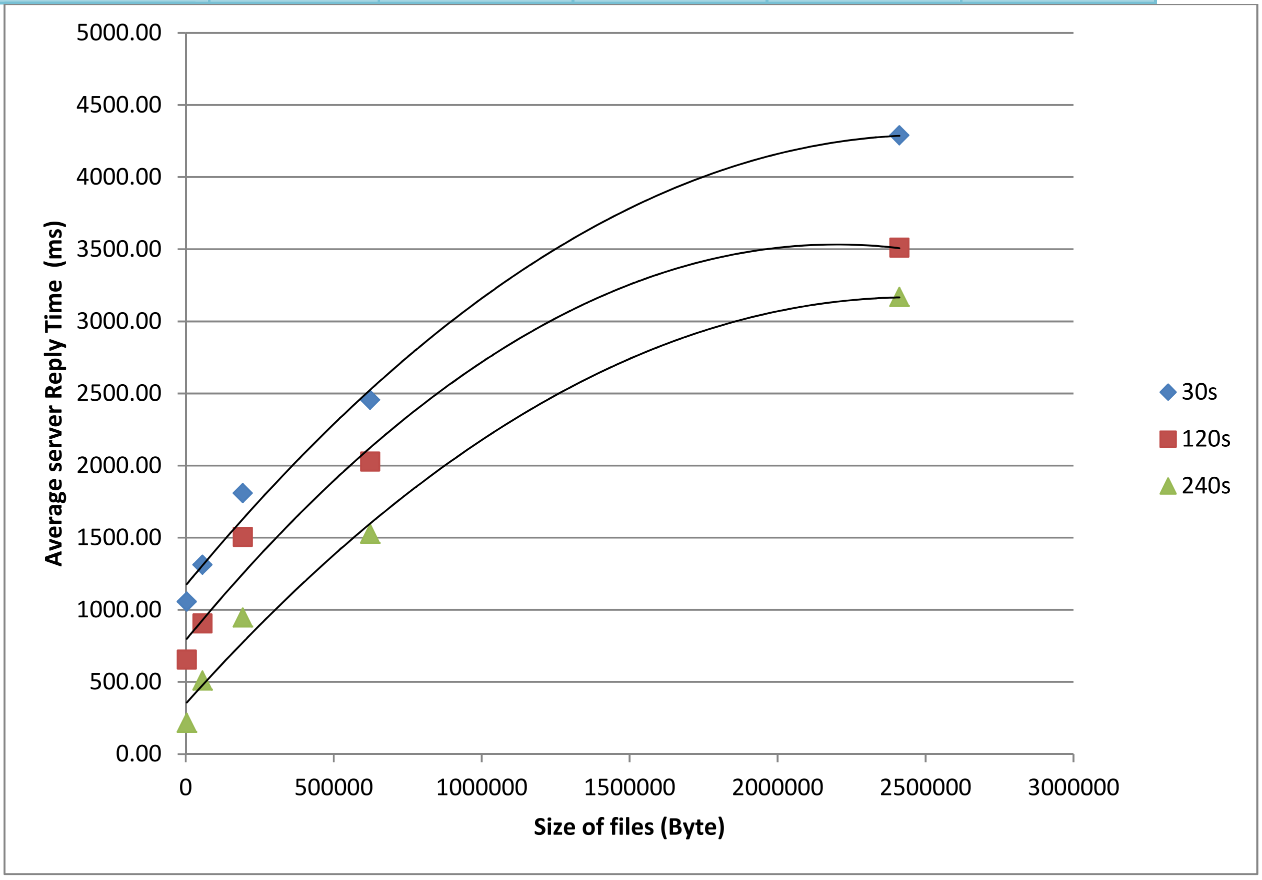


Figure 7-23: Average Server Reply Time for fixed size of HTTP and PSIRP requests

7.2.3.4 Server Response Time

The following table and figure show the average server response tine with different fixed size of files in several testing period.

Table 7-18: Average Server Response Time Fixed size of HTTP and PSIRP requests

Average Server Response Time (ms)	3KB	55KB	188KB	608KB	2.3MB
30s	1058.7148	1315.3495	1811.0654	2458.3509	4294.0000
120s	655.8552	907.2166	1507.1826	2029.7268	3513.9755
240s	217.8586	510.1381	948.19	1528.93	3170.9972

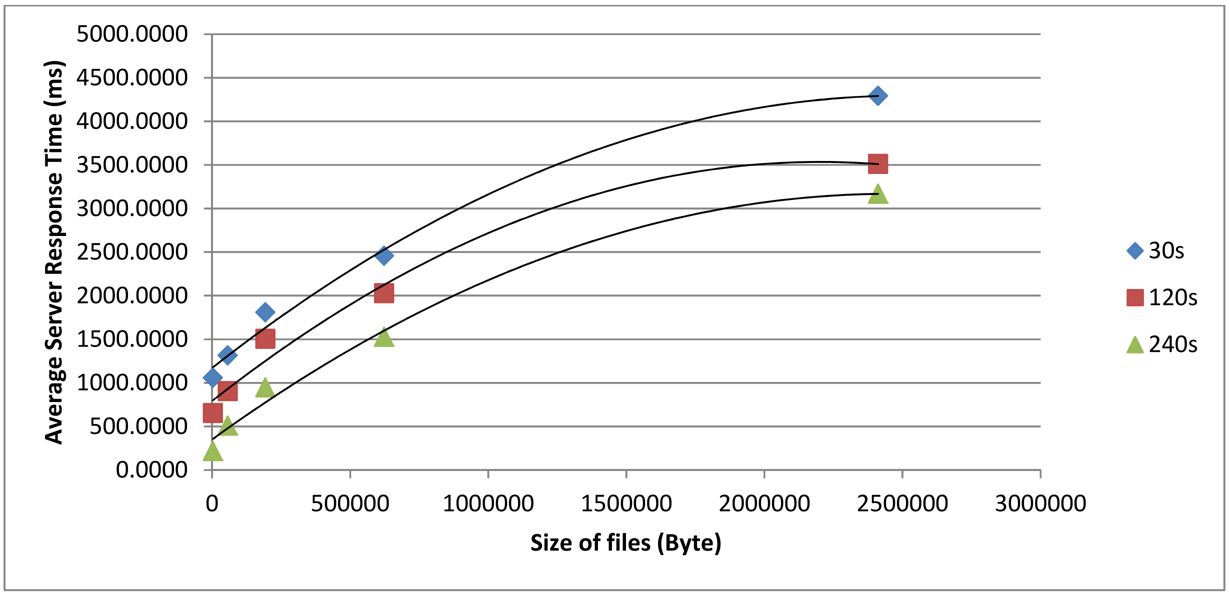


Figure 7-24: Average Server Response Time Fixed size of HTTP and PSIRP requests

7.3 Summary

In this chapter, we evaluated the system with different metrics. The results show that our system could satisfy the clients' requirement. It successfully fetched both HTTP contents from web server and PSIRP contents from PSIRP Overlay. According to the test results, we summaries several conclusions listed below:

1. Under the same condition, smaller files have a higher document hit ratio.
2. The average request response time is influenced by the Hit Ratio. For a given testing time and number of URLs, a higher hit ratio means more content can be retrieved from the local cache. Such a cache hit saves the whole period of data request and retrieval process between the proxy and remote server. The higher the hit ratio is, shorter the RRT is.
3. The RRT increases when the file size increases. Larger files needs more time to be retrieved from a remote server to the client or transmitted from a publisher to subscriber.
4. A higher DHR leads to a lower average server reply time, because more contents are fetched from the local cache. The average server reply time depends on the traffic load on the path and the performance of remote server or publisher proxy in the PSIRP overlay.
5. The larger the file size, the longer the proxy connect time.
6. The initial cost for HTTP + PSIRP request (0KB) is 1129ms. The request rate is 0.89

8 Conclusions and Future work

In this chapter, we will firstly state the conclusion of this project. Then we will discuss the limitation of our system and list the possible future work. The last subsection describes some reflections on social, economic, and ethical issues associated with this thesis project.

8.1 Conclusions

This thesis project's goal was to build a publish/subscribe network and use it to design, implement, and evaluate a proxy for HTTP. The proxy includes two basic functions. One function is to act as a normal HTTP proxy with local cache. The other function is that the proxy provides a service that allows client query a publication through publish/subscribe network with PSIRP URL. We implemented the BlackAdder prototype on Linux operating system and used Java as our programing language. The application programming interface provided by PURSUIT was used in this program to exploit the functionalities of publish/subscribe networks. In order to achieve the caching function of the proxy, we utilized the Couchbase Server. From the Couchbase server, we could monitor the cache status and request traffic. Compared to the pub/sub HTTP proxy developed by PURSUIT project, our proxy has a different structure and provides different functions. The PURSUIT HTTP proxy the proxy simply transforms HTTP request to a pub/sub request and forwards it to the BlackAdder Testbed ICN or reads the pub/sub requests and converts it to HTTP format in order to send it to a remote server. Both the BlackAdder ICN and PSIRP overlay provide publish/subscribe functions. However, the former always converts the format of request, and then subscribes to the content from the pub/sub network. So the proxy needs to wait for an existing publisher to respond or time out. Our system allows a client to send both HTTP and PSIRP requests. Because the proxy can identify the type of request and it has ability to use different threads to process the requests it can support both types of requests and can make content returned from HTTP requests into pub/sub content.

The client has two options to obtain services or resources: sending an http://… or a psirp:// request to the proxy via different ports. The proxy utilizes two socket servers in order to listen to these two ports and to process the different incoming requests. The HTTP socket server provides the following services: When the proxy receives a HTTP request from a client, it will first search its local cache. The proxy will return the requested content to client if the item is stored in the local cache. Otherwise, the proxy forwards the request to the web server. When the proxy receives the information from a remote server, it forwards it directly to the client and saves copy of the content in its cache. One of the most important abilities of this proxy is to generate and cache PSIRP content. The proxy generates PSIRP content and publishes this content to the PSIRP overlay. Once this publication is subscribed to by other clients in the overlay, the proxy will transmit the publication to the clients who express and interest. The PSIRP socket server parsew psirp URLs. The extracted Rendezvous Identifier (RID) and Scope Identifier (SID) can be utilized to searching for content in the local cache and if not found to subscribe for this PSIRP content via the PSIRP overlay. The system creates BlackAdder Events to start publishing data, stop publishing data, start publishing scope, and to transmit data. A Subscriber Evert Handler is responsible for receiving published data from the overlay and forwarding the content to clients.

We built a publish/subscribe network and tested the functions of our system while varying several parameters. The results showed that the proxy could achieve the targets we set. The proxy could satisfy the proposed requirements of clients. In the final part of this project, we evaluated the system through two aspects: from the client’s point of view and from the proxy’s point of view.

We measured the system’s usability and performance from the client’s point of view. The general performance parameters include document hit ratio, byte hit ratio, average request response time, and request quantity (number of HTTP requests and PSIRP requests). For certain number of optional HTTP and PSIRP resources, we saw the Average Document Hit Ratio (HDR) and Byte Hit Ratio (BHR) show an increasing trend as the testing time increases. If the testing time is same, the Average DHR and BHR show a decreasing trend as the total number of URLs increases from 100 to 500.The caching strategy could be changed based on the requirements of clients and the local storage capacity. The average request response time is influenced by the Hit Ratio. For a given testing time and number of URLs, a higher hit ratio means more content can be retrieved from the local cache. Such a cache hit saves the whole period of data request and retrieval process between the proxy and remote server. The higher the hit ratio is, shorter the RRT is. Moreover, we evaluated how the transfer size of a TXT object influences the performance of the system. We utilized different sizes of object as a file resource. The test results showed that smaller files have a higher document hit ratio. This occurs because within a certain period and with a stable Internet data rate, the transmission ability of network is also stable. No matter how big the file is, the network can only handle transfer traffic at this data rate. Usually, there is a size limit on the size of cached files. This limit prevents users connected via slow connection, from storing large images and videos in the cache. The cache strategy should be based upon the requirements of the clients. The RRT increases when the file size increases. Larger files needs more time to be retrieved from a remote server to the client or transmitted from a publisher to subscriber. Within a limited period of time, clients could request less big size files compared to smaller size files scenario.

We also analyzed the network’s performance from the Proxy’s point of view. We evaluated the proxy’s performance based upon computing the connect time and reply time for communicating between the client and proxy, as well as between the proxy and a remote server. The server’s response time equals the proxy’s connect time plus the server’s reply time. The test result indicates that the average server reply time increases dramatically when the number of optional TXT files increases. A higher DHR leads to a lower average server reply time, because more contents are fetched from the local cache. The average server reply time depends on the traffic load on the path and the performance of remote server or publisher proxy in the PSIRP overlay. Furthermore, as indicated in the experiment, the larger the file size, the longer the proxy connect time. With the same network circumstances and transmission rate, the remote server needs more time to return larger files to the proxy.

In conclusion, a publish/subscribe proxy for HTTP based communication has been implemented. The system supports clients sending both PSIRP and HTTP requests. Each proxy connects to the Internet and the PSIRP overlay. The PSIRP overlay stores and maintains pub/sub contents. This proxy can be developed by others who are interested in this topic or can be utilized as a module in the PURSUIT project.

8.2 Future work

In this thesis, when we built the Publish/Subscribe network, we used a node/link –local strategy so that the operation of the rendezvous function is similar to service discovery solutions. There are other rendezvous strategies such as domain-local and global rendezvous strategies which are more complicated. This implement could be further improved to better suit a real networking environment. Secondly, our system has the ability to transmit TXT files and the evaluation mainly focused on TXT files. Additional content types should be supported by the proxy, such as video and voice services. Last but not least, in order to improve the document hit ratio and byte hit ratio, more effective cache strategies could be deployed, depending on requirements of the clients.

8.3 Required reflections

The project intends to discuss the possibility of a hybrid proxy which combined the HTTP and PSIRP functions. In order to accommodate for the current Internet structure and Information-centric Networking architecture, our system provides the ability that the users may benefit from obtaining HTTP contents and publications. The purpose is to foster the introduction and spread of content based access.

A proxy provides multiple functionalities including content-filtering, caching, accessing services anonymously, and censorship, etc. The content-filtering feature protects the security of users' computer and prevents hackers from damaging the hardware and software. However, the anonymous feature provides the possibility that allows the hackers hide their real IP, which increases the difficulty of network security and privacy. One ethical concern of our proxy is that it may benefit both trustful users and for the people who probably will our system to do illegal activities such as data theft.

The Pub/Sub proxy enables clients to have the advantages of publish/subscribe network. The publish/subscribe model decouples publisher and subscriber in time, space, and synchronization. Publisher declares the topics on which they intend to publish and subscriber register to the topics of interest. The new pub/sub internetworking architecture will restore the balance of network economics incentives between the sender and the receiver. Our proxy is responsible for publishing contents to pub/sub overlay. The content-filtering feature becomes even more crucial because publish/subscribe network is based on trust mechanism. While the scoping function of PSIRP could help the system accomplish access control. The establishment of our system provides a bridge that connects the current Internet structure and Information-centric Networking architecture, which would give clients and other developer benefits.

References

[1] R. Moskowitz and P. Nikander, “Host Identity Protocol (HIP) Architecture,” Internet Request for Comments, vol. RFC 4423 (Informational), May 2006.

[2] K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” Internet Request for Comments, vol. RFC 2474 (Proposed Standard), Dec. 1998.

[3] Sasu Tarkoma, Mark Ain, and Kari Visala, “The Publish/Subscribe Internet Routing Paradigm (PSIRP): Designing the Future Internet Architecture,” the Future InternetA European Research, vol. 1, no. 37, pp. 102–111, 2009.

[4] Nikos Fotiou and Dirk Trossen, “Illustrating a Publish-Subscribe Internet Architecture,” 2010. [Online]. Available: http://www.psirp.org/. [Accessed: 22-Aug-2012].

[5] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica, “ROFL,” ACM SIGCOMM Computer Communication Review, vol. 36, no. 4, p. 363, Aug. 2006.

[6] T. Berners-Lee, R. Fielding, and H. Frystyk, “Hypertext Transfer Protocol – HTTP/1.0,” Internet Request for Comments, vol. RFC 1945 (Informational), May 1996.

[7] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica, “A data-oriented (and beyond) network architecture,” ACM SIGCOMM Computer Communication Review, vol. 37, no. 4, p. 181, Oct. 2007.

[8] Vu, Tam, Akash Baid, Yanyong Zhang, Thu D. Nguyen, Junichiro Fukuyama, Richard P. Martin, and Dipankar Raychaudhuri. “DMap: A Shared Hosting Scheme for Dynamic Identifier to Locator Mappings in the Global Internet.” In Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems, 698–707. ICDCS ’12. Washington, DC, USA: IEEE Computer Society, 2012. doi:10.1109/ICDCS.2012.50.

[9] Dmitrij Lagutin, Kari Visala, and Sasu Tarkoma, “Publish/Subscribe for Internet: PSIRP Perspective,” Valencia FIA book, vol. 4, pp. 75–84, 2010.

[10] Dirk Trossen (ed.), George Parisis, Borislava Gajic, Janne Riihijarvi, Paris Flegkas, Pasi Sarolahti, Petri Jokela, Xenofon Vasilakos, Christos Tsilopoulos, Somaya Arianfar, and Martin Reed, "Deliverable 2.3: Architecture Definition, Components Descriptions and Requirements", Publish Subscribe Internet Technology (PURSUIT) project, Deliverable D2.3 Available at http://www.fp7-pursuit.eu/PursuitWeb/?page_id=158, October 2011.

[11] George Parisis, Kari Visala, Borislava Gajic, Janne Riihijarvi, Paris Flegkas, Pasi Sarolahti, Petri Jokela, Xenofon Vasilakos, Christos Tsilopoulos, and Somaya Arianfar, “Deliverable 2.2:Conceptual Architecture: Principles, patterns and sub-components descriptions,” Publish Subscribe Internet Technology (PURSUIT) project, Deliverable D2.2 Available at http://www.fp7-pursuit.eu/PursuitWeb/?page_id=158, May 2011.

[12] C. Tsilopoulos, D. Makris, and G. Xylomenos, "Bootstrapping a Publish/Subscribe Information Centric Network," to appear In Future Network and Mobile Summit 2011, 2011.

[13] Petri Jokela, Janne Tuononen, Jimmy Kjallman, Pekka Nikander, Jari Keinanen, Andras Zahemszky, Dirk Trossen, Borislava Gajic, George Xylomenos, and Dmitrij Lagutin. Progress Report and Evaluation of Implemented Upper and Lower Layer. Publish-Subscribe Internet Routing Paradigm FP7-INFSO-IST-216173, June 30, 2009. http://www.psirp.org/publishcations.html.

[14] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander, “LIPSIN,” ACM SIGCOMM Computer Communication Review, vol. 39, no. 4, p. 195, Aug. 2009.
[15] Arto Karila, "Publish/Subscribe Internetworking Special Course, Evolution vs. Revolution" wiki.fp7-pursuit.eu/uploads/f/fb/02a-Evolution_vs_Revolution.ppt. Sep. 2011.
[16] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. “Rofl.” ACM SIGCOMM Computer Communication Review 36, no. 4 (August 11, 2006): 363.
[17] Paul J. Leach, Tim Berners-Lee, Jeffrey C. Mogul, Larry Masinter, Roy T. Fielding, and James Gettys. “Hypertext Transfer Protocol -- HTTP/1.1”, n.d. http://tools.ietf.org/html/rfc2616.
[18] Wikipedia contributors. “Web Cache.” Wikipedia, the Free Encyclopedia. Wikimedia Foundation, Inc., August 9, 2012. http://en.wikipedia.org/w/index.php?title=Web_cache&oldid=506602432.
[19] Landon Curt Noll, ‘FNV Hash’, FNV Hash, 14-May-2012. [Online]. Available: http://isthe.com/chongo/tech/comp/fnv/. [Accessed: 16-February-2013].
[20] Kevin Kaichuan He. “Kernel Korner: Why and How to Use Netlink Socket.” Linux J. 2005, no. 130 (February 2005): 11–.
[21] "Blackadder Application Programming Interface", Available at https://github.com/fp7-pursuit/blackadder. Last Access: Jan, 2013
[22] Jimmy Kjallman, Nikos Fotiou, Borislava Gajic, Dmitrij Lagutin, YiChing Liao, George Parisis, Charilaos Stais, Dimitris Syrivelis, and Christos Tsilopoulos. Progress Report of Component Implementations. Publish-Subscribe Internet Technology FP7-INFSO-ICT-257217, March 05, 2012. http://www.fp7-pursuit.eu/PursuitWeb/?page_id=158.
[23] Petri Jokela, Janne Tuononen, Teemu Rintaaho, Jukka Ylitalo, Dirk Trossen, Dmitrij Lagutin, Janne Riihijärvi, George Xylomenos, and Jimmy Kjallman. Implementation Plan Based on Conceptual Architecture. Publish-Subscribe Internet Routing Paradigm FP7-INFSO-IST-216173, September 30, 2008. http://www.psirp.org/publishcations.html.
[24] George Parisis (ed.), Dirk Trossen, Ben Tagger, Christos Tsilopoulos, Dimitris Syrivelis, Martin Reed, Mays Al-Naday, Vassilios Vassilakis, Jimmy Kjallman, and Borislava Gajic, ‘Deliverable 3.4: Integration and Demonstration Plan’, Publish Subscribe Internet Technology (PURSUIT) project, D3.4 Available at http://wiki.fp7-pursuit.eu/uploads/f/f6/INFSO-ICT-257217_PURSUIT_D3.4_Integration_and_Demonstration_Plan.pdf, April 2013.
[25] Janne Riihijärvi, Borislava Gajic, George Parisis, Ben Tagger, Dirk Trossen, Konstantinos Katsaros, Alexandros Kostopoulos, Giannis F.Marias, and Nikos Fotiou. First Report on Qualitative and Quantitative Architecture Validation. Publish-Subscribe Internet Technology FP7-INFSO-ICT-257217, January 05, 2012. http://www.fp7-pursuit.eu/PursuitWeb/?page_id=158.
[26] Alex Rousskov,Valery Soloviev. "A performance Study of the Squid Proxy on HTTP/1.0"
[27] Wikipedia contributors. “Packet loss.” Wikipedia, the Free Encyclopedia. Wikimedia Foundation, Inc., Feb 11, 2013. http://en.wikipedia.org/wiki/Packet_loss
[28] CCNx Protocol Organization Website. Available at: http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html. Last Access: April, 2013

TRITA-ICT-EX-2013:68