Arseny Kurnikov

Web browser for delay-tolerant networks

Faculty of Electronics, Communications and Automation

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 29.3.2011

Thesis supervisor:

Prof. Jörg Ott

Aalto-yliopisto
Teknillinen korkeakoulu

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author: Arseny Kurnikov

Title: Web browser for delay-tolerant networks

Date: 29.3.2011 Language: English Number of pages: 5+66

Faculty of Electronics, Communications and Automation

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor and instructor: Prof. Jörg Ott

Due to growth of the Internet, the number of devices increasing and the structure of networks becoming more complex, the problem of time delays during information transmissions has arisen. In environments with long transmission delays modern protocols may become inefficient or even useless. Delay-tolerant Networking (DTN) is one approach that allows to solve the problem of long transmission delay times.

In the thesis, an approach to web access in such networks is proposed. The problem of data transmission in the networks with long delays is considered. Special methods exist for data transmission in computer networks. But traditional data transmission protocols do not work well in networks with long delays, e.g. when transmitting over long distances, such as in space, or when connectivity may be disrupted, such as in mobile networks. It is necessary, therefore, to replace TCP and to change the existing web protocol (Hypertext Transfer Protocol – HTTP) in order to allow HTTP data transmissions in DTN environments.

In the thesis, HTTP is analyzed and an adaptation of HTTP to DTN environments, as proposed in earlier research, is reviewed and extended further. A client part is created and the implementation is described. The client allows solving the problem of HTTP over DTN usage. An open-source browser is modified and the necessary extensions are developed. The extensions allow to use the DTN transport protocol (i.e. the Bundle Protocol) as another option of transport other than TCP.

The software module for a web browser is built on the Mozilla platform. It was shown that it is possible to create a browser to work in DTNs.

Keywords: Delay-tolerant Networks, Browser, HTTP

Preface

I would like to thank prof. Jörg Ott for the help and guidance during the work, and William Martin for proofreading the final version of the thesis manuscript. I would like to thank Teemu Kärkkäinen for providing a DTN-enabled web server.

Arseny Kurnikov Otaniemi, 29.3.2011

Contents

Abstract **ii**

Preface **iii**

Contents **iv**

Symbols and abbreviations **v**

1 Introduction **1**
1.1 Problem Statement 1
1.2 Objectives and Scope 2
1.3 Outline 3

2 Background **4**
2.1 Related Work 4
2.2 Main Principles of Data Networks 5
2.3 Network Architecture 6
2.4 Summary 10

3 HTTP **11**
3.1 HTTP and the Web 11
3.2 HTTP Protocol Overview 13
3.3 Request and Response Headers 22
3.4 Authentication Mechanisms 27
3.5 HTTP Characteristics and Limitations 29
3.6 Summary 30

4 Delay-Tolerant Networks **31**
4.1 Data Transmission Problems 31
4.2 DTN Architecture and Services 33
4.3 HTTP over TCP and DTN 35
4.4 Summary 44

5 Browser DTN Extension **45**
5.1 Browser Structure 45
5.2 Browser Extension 48
5.3 Proxy Implementation 54
5.4 Extension Testing 57
5.5 Summary 60

6 Conclusions **62**

References **64**

Symbols and abbreviations

Abbreviations

AJAX	Asynchronous JavaScript and XML
ARP	Address Resolution Protocol
ASN.1	Abstract Syntax Notation One
DNS	Domain Name System
DTN	Delay-tolerant Networking
DTNs	Delay-tolerant Networks
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPC	Inter Process Communication
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
MAC	Media Access Control
MIME	Multipurpose Internet Mail Extensions
NASA	National Aeronautics and Space Administration
NFS	Network File System
OSI	Open System Interconnection (basic reference model)
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
PPP	Point-to-point protocol
RIP	Routing Information Protocol
RPC	Remote Procedure Call
RTSP	Real Time Streaming Protocol
SCTP	Stream Control Transmission Protocol
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XDR	External Data Representation
XML	eXtensible Markup Language
XPCOM	Cross Platform Component Object Model

1 Introduction

In recent times an active development of the Internet and its introduction in almost all areas of people's everyday life has resulted in the main problems in this direction being the problems of data transmission channels' capacities and also their reliability.

Reliability, that is the probability of messages' delivery to a destination, is one of the key quality features of a computer network. But there are situations where the probability that a message reaches its destination is very low. For example, consider the situation of a technological catastrophe. In cases where there is no technical infrastructure, a signal simply will not get to the nearest node of the network and an attempt to communicate over another route when the response time exceeds a particular threshold will result in the failure of the whole communication session. Apart from this situation others are also possible, when the response time is very large.

The situation when users move introduces the problem of links disruptions. The mobility of the users leads to the fact that the links are not permanent and delays vary greatly. There are disconnections when a user moves out of range of one base station and should be served by another base station. Thus, any data transmission may be interrupted at anytime and it is unpredictable when it can be resumed.

Another example of this phenomenon is observed during space and satellite communications. In these cases, signal transmission delays may reach several minutes, hours or even days. The usage of traditional data transmission methods gives users no opportunity to wait for a server's response.

To deal with these situations new data transmission methods should be developed that are tolerant to such lengthy and unpredictable time delays. One such approach is called Delay-Tolerant Networking (DTN). It allows communicating when no end-to-end path exists or the time delays are significant. The DTN development is on early stage, with basic concepts and specifications available, yet many are still to come in this new research field.

1.1 Problem Statement

DTNs are networks that are based on other working principles than traditional networks that make them capable of dealing with situations when time delays are large. The main principle is to "store-and-forward". It means that during the message exchange, nodes of a network store messages in case when there is no link to the next node. The time of storing a message may be in order of minutes, hours or even days until the link goes up and then the message can be transmitted further.

The development of DTNs evolves from the Interplanetary Internet when the Internet communication principles were brought to deep space communications. It was shown that the traditional protocols fail when delays are long and there are disruptions that occur from time to time. This situation can also be observed in mobile networks. When a user moves out of the coverage of one base station and is not yet served by the other one this is a disruption. This suggests using DTN concepts to support mobile Internet access.

It is known that the Internet for most users is strongly associated with Web-surfing. Therefore, the development and realization of a protocol for DTNs to work with Web-browsing (HTTP over DTN) would be useful. A server part of the HTTP chain was done as a research project at Aalto University [1]. A browser that would be able to work in DTNs is, nevertheless, still missing. HTTP features do not allow using it in DTNs "as is" and require some additional work to be done in order to implement its functionality in delay-tolerant environments. In this thesis, a detailed analysis of the problem is performed and a solution is proposed which is implemented in an open-source browser by introducing DTN feature as an alternative to the main TCP protocol.

The problem of HTTP in DTNs is that HTTP is a conversational protocol, i.e., a lot of messages have to be transmitted between a browser and a server in order to fetch a web page because it usually comprises many resources and all of them are retrieved one by one..

TCP does not work in DTNs either. TCP is based on the assumptions that do not hold in DTNs. One of these assumptions is the existence of an end-to-end path between communicating nodes. Moreover, TCP performance heavily depends on short transmission delays along the path [2]. When delays are long, or there is no infrastructure that provides permanent connectivity between nodes, TCP fails.

The problem addressed in this thesis is to adopt HTTP for usage in DTNs and to allow a web-browser to communicate over a DTN[1] along with the TCP transport option.

So far, only proxy-based solutions for DTN-based web access were proposed, but those are naturally limited because a proxy needs to perform complete web page evaluation as a web browser does and a web browser can provide better feedback to a user when it knows about DTN-based web access. This thesis takes a step forward to fill this gap.

1.2 Objectives and Scope

This thesis is dedicated to the research in a new area of information transmission – DTN and the usage of HTTP in these networks. The goal of this thesis is to create a DTN-enabled web-browser to allow the intrinsic support for DTN-based communications beside TCP. This task is twofold:

1. TCP should be replaced with the DTN protocol as the means of transport.

2. HTTP has to be modified to work in delayed environments. The modification has to be implemented in a browser retaining HTTP's main features.

The issues that arise during the work are related to the problems that are solved by the DTN protocol and the problems that HTTP introduces when using it in DTN environments. The focus of the work is on the client side of HTTP. There is

[1]Here a DTN is a Delay-tolerant Network. We use the abbreviation DTN for both Delay-tolerant Networking and Delay-tolerant Network. The actual meaning is derived from the context.

a server implementation that can be used along with the client that is developed in the thesis [1].

A proxy was developed that receives browser requests over TCP, constructs a bundle with the request to the DTN server, receives a response from it, saves the resources to the cache and waits for the sequential requests to send these cached resources back.

But the proxy is only a partial solution of the "HTTP over DTN" problem. The progress in the DTN research field becomes possible only when a browser that is able to work over DTN appears. The development of the browser is the goal of this work.

A modified browser to work in DTNs has to be developed and implemented. The thesis includes the detailed description of TCP networks and the HTTP protocol. In particular, HTTP should be analyzed and modified to work in DTNs. The author will perform a detailed problem analysis and the solution will be provided.

Additionally, a proxy shall be developed. The proxy allows to use the DTN-enabled web-browser in conjunction with non-DTN web servers.

1.3 Outline

The approaches that were used in this research include the analysis of the existing protocol, its structure and main parts. And based on this analysis, an extension to an open-source web-browser is created. The main idea behind the modification of the protocol is to pack resources into packets and to send them at once.

In Chapter 2, the principles of modern networking technologies are described. This is done to compare these principles to the DTN concepts that are discussed later in Chapter 4. In Chapter 3, HTTP is discussed. The main problems that do not allow to use it in DTNs are pointed out. In Chapter 5, the implementation of the browser extension and the proxy is described. Finally, in the last chapter, conclusions about the work are made and possible further research topics are introduced.

2 Background

In this chapter, the basics of networks structure and their functioning are described. The importance of standards is discussed. Protocols that are used in traditional networks are introduced, most of them being shown not to work properly in DTNs.

A client-server architecture is described. This architecture defines the main principles that are used in modern networks. In particular HTTP is based on them.

In the first section of the chapter, other DTN related projects are introduced.

2.1 Related Work

The main ideas of the HTTP over DTN usage are introduced in [3]. The server part implementation is provided in [1]. The approach that is proposed in these works is to aggregate HTTP resources and send them all at once. In this section, we will look at other projects that are related to DTNs and HTTP.

The Networking for Communication Challenged Communities (N4C) project is looking at ways to extend Internet access to remote regions that do not have reliable network access today [4]. There are many locations in the world that cannot be reached by conventional links like optical fibres, copper cables, radio waves or satellite links. These regions are sparsely populated and spread over large areas. It is hardly possible that conventional technologies can be economically deployed there.

However existing transport and people movements in these areas provide the opportunities to carry data. The goal of the N4C project is to allow networking in communication challenged communities [4]. DTN plays a key role in N4C and N4C is extending DTN so that it can integrate with modern Internet applications and provide capabilities similar to original applications based on traditional networking technologies.

One application of DTNs is deep space communications. As an example of what kind of problems arise when a signal goes to space satellite communications can be taken into consideration. For example, delays of geostationary satellites during inter-continental calls disturb. But delays for applications like video-conferencing make them useless and when using some of protocols it can result in total chaos. Satellites are relatively close to the Earth. If a signal has to go further delays become even longer. Problems that are encountered in this setup are the ones that are solved by DTN [5].

Postellation is an implementation of DTN. Its features are a lean protocol implementation that is well-suited for embedded systems; a smart HTTP proxy that allows web application developers to use DTN “transparently”; and easy deployment of DTNs to attract more users to DTNs, develop a community [6].

Another project that deals with the situations when there is no permanent connectivity exists is the Drive-thru Internet [7]. Its goal is to study the possibility for mobile users to access the Internet while moving in vehicles. The idea of the project is to provide some areas within the highway where the wireless network connectivity is available for some short period of time. Usually wireless networks are used on a

stationary basis. In the Drive-thru Internet this technology is applied to a rather unusual scenario: to provide the network access for fast moving vehicles.

The intermittent connectivity is considered to be the property of the access. To provide some "useful" connectivity the protocol that is developed within the project (Persistent Connection Management Protocol) assumes that a link may break at any time. On top of these periods of available wireless access a session is maintained by this protocol. Thus network applications can work in this environment as it was a usual "always-on" network.

In [8] the authors propose to use HTTP as a transport protocol for different kinds of data in DTNs. HTTP is considered to be a solution to transfer data on a hop-by-hop basis. The main idea is to use MIME to describe the contents of the DTN packets. MIME (Multipurpose Internet Mail Extensions) is the specification that originally was developed to describe different types of content in e-mail messages.

The adopting of the HTTP protocol for DTN was proposed in [8]. In particular it says:

> "A simple approach leverages existing standards, using the HTTP as a transport-layer-independent 'session layer' between each two communicating delay-tolerant networking (DTN) nodes" [9]. "Rather than an end-to-end communication across a path as in the traditional Internet, this use of HTTP is hop-by-hop between communicating peers. HTTP runs over its 'convergence layer' – that is, a transport layer – suitable for local network conditions between each HTTP-capable node. New `"Content-Source:"` and `"Content-Destination:"` headers are added to HTTP."[8]

In this approach it was proposed to use series of HTTP transfers. Each transfer is performed between two neighbor nodes. At last an object gets to the destination node.

It should be noted here that this approach does not deal with web-browsing in its original sense. There are no servers and clients. It may be compared to a P2P network where users share resources. Although this approach to HTTP in DTNs is quite interesting, the work in this thesis is focused mostly on the developing of a browser that is able to work with a DTN-enabled web server.

2.2 Main Principles of Data Networks

The Internet is a world-wide phenomenon. The network that previously was used by a limited number of scientists, public servants, and educational institutions in their professional activity became available for large and small companies and for individual users.

A computer nowadays is not a stand-alone device for data processing but "a window" to computer networks, a tool for communication with network resources and other network users. The main goal of computer networks is the shared usage of the resources and interactive communication within one company as well as outside of

this company. Resources are data, applications and peripheral devices like printers, modems etc., and services like e-mail, WWW, FTP and so on.

When local networks achieved the high level of development and the volume of information transmitted increased they became the components of larger networks. Data that is transferred according to one of the possible routes from one local network to another is called routed data. Protocols that support the data transmission between networks along several routes are called routing protocols [2].

The Internet is a world area network that covers the whole world [10]. It grows by 7-10% each month [11]. Internet is a core that provides a communication medium for different information systems of different companies all across the world. Earlier this network was used only as a medium for files transfer and electronic mail purposes. Today more complicated tasks of shared access to the resources are solved.

When connecting several computers into a network there are additional difficulties that appear. The main one is the selection of the way to organize physical lines that communicate computers. This is called a network's topology. The choice of the topology greatly depends on parameters. Existing topologies are the basis of modern local area networks (a mesh, a star, a tree, a bus and a ring).

Channels[2] can be wired and wireless. A channel is characterized by the values that are important from the point of view of the network organization: the data transmission speed, the maximum reach of the channel, the interference robustness, the price.

In this work, the problem of data transmission speed is considered. This parameter is one of the critical parameters when transmitting data. The problem comes from the fact that when the speed is low there is a huge delay and the communication with a server becomes impossible.

2.3 Network Architecture

Recently, computer networks are widely used in all areas of science and industry. A network is a collection of objects that are devices to transfer and process data. At the moment, the International Standard Organization has defined a network as a serial bit-oriented data transfer between independent devices linked one to another [12].

It is known that the following elements are usually included in a network: network computers (with one or more network cards), communication channels (wires, radio channels, optical channels, etc.), signal transformers of different kind, various network equipment.

There are two notions of a network:

– a communication network, and

– an information network.

[2]We speak about links and paths in the Internet. A channel is an abstraction that is mainly used in wireless technologies. Here, though, we use the word "channel" in a sense that it is a physical media to transmit data.

A communication network is designated to transfer data. It is used for the tasks of data transformation, too. Communication networks differ depending on the type of physical communication facilities. An information network is designated to store, process and exchange information. It consists of information systems. On the base of a communication network a group of information networks may be built. The World Wide Web is an example of an information network.

An information system is a system that acts as both a provider and a consumer of information. An information system is an object that is able to store, to process and to transfer data. An information system includes: computers, software, users and other parts that are used in data processing and transmission. An information system that is used to solve a user's tasks is called a client. A client is a computer that has a network adapter, a communication channel and network software.

Networks transmit information in blocks of data according to the communication procedures between objects. These procedures are called data transmission protocols. A protocol is a set of rules stating a format of data and procedures of information exchange between two or more devices.

A client-server architecture is a concept of information networks where the main part of resources is located on servers which serve their clients. This architecture has two types of components: servers and clients [2].

A server is an element that provides some service to other objects in a network according to their requests. A service is a process of handling clients. A server works on client requests and controls the process of requests execution. After performing each task a server sends results to a client, issued the given request. A service function in the client-server architecture is described by a set of applications which perform different application processes.

A process that calls a service function with the help of some operations is called a client. Clients are workstations that use a server's resources and provides convenient user interfaces. User interfaces describe procedures of a user's interaction with a system or a network. A client initiates the interaction and uses the server's services (e.g. e-mail). During this process a client requests a type of service, establishes a session, receives the necessary results and drops the session [2].

Problems of Network Standardization

Along with the selection of a topology it is necessary to solve a set of problems. Altogether these problems (the choice of topology, physical and electrical characteristics of a network, the computers addressing system and rules and a format of data transmission) are called a networking technology. The number of networking technologies developed would lead to the networks incompatibility if there was no research into these technologies within the international standardization organizations. Nowadays some standard networking technologies won the main part of the market, for instance, Ethernet, Token Ring. Others are not wide-spread but they have, nevertheless, played an important methodological role in the development of networks. And yet the others are quite perspective but are not wide-spread due to many reasons.

In the field of DTN the main entity that works on protocol standards is the DTN working group [13].

International Standards

To make it easier to develop, analyze and implement the procedure of messages exchange between users and applications that work on different computers this procedure is decomposed into several hierarchically connected tasks. It is called a layered approach.

When transmitting messages both participants of a network exchange have to make several agreements. For example, they have to agree upon the levels and the form of electrical signals, the way of determining message sizes, the methods of the error control. In other words, the agreements have to be accepted on all the layers starting from the lowest one where bits are transmitted and ending with the top layer that details how the information is to be treated. These formalized rules that define the sequencing and the format of messages are called protocols. The messages are exchanged by networking components of different nodes but on the same layers.

A set of protocols that is hierarchically organized and solves the task of nodes communication is called a stack of communication protocols.

The protocols of neighboring layers of the same node interact with each other according to well-defined rules, too. These rules are called an interface. An interface defines a set of services that are provided to the higher layer by the lower one.

The International Standards Organization (ISO) developed a model that clearly defines different layers of systems interactions, giving these layers standard names and points to the work that has to be done by each layer. This model is called the Open System Interconnection model (OSI) [12]. The Internet Engineering Task Force (IETF) developed the TCP/IP model (the Internet protocol stack) that has fewer layers and is more practical when talking about the Internet [14].

In the DTN working group that solves the problems of data transmissions in delayed environments the standards are being developed. They are also used in this work.

Protocol Stack Model

To unify data representation in networks with heterogeneous hardware and software the layered model is used. This model describes rules and data transmission functions in different network environments during a communication session. The main components of the model are layers, application processes and physical communication devices.

Each layer accomplishes its own task during the process of data transmission across a network. OSI defines 7 layers of functionality each of them serving different parts of open systems communications. The Internet protocol stack follows the same model but it is simplified having only 5 layers of functionality [2].

Physical layer. Describes an environment of data transmission (cable, radio or optical fiber), the physical characteristics of the environment and data transmission

principles (channel multiplexing, modulation methods, synchronization parameters, maximum distance and time of reply awaiting).

Link layer. Describes how data packets are transferred including coding methods (bit sequences that mark the beginning and the end of a packet). E.g. Ethernet in the header fields defines which host or hosts in a network the packet is designated to.

Internet layer. At the beginning it was designed to transfer data from one (sub)-network to another. Examples are X.25 and IPC in ARPANET. With global network development additional possibilities were introduced into this layer that include the possibility of data transfer from one network to another independently from the lower layer protocols and also the possibility to request data from a remote host, for instance, in ICMP (that is used to transfer the diagnostic information about an IP-network) or IGMP (that is used to control multicast flows). The IP protocol packets may include the code that defines which protocol on the next layer to use to get data from the packet. This code is a unique IP-protocol number.

Transport layer. Protocols may solve the task of reliable data transfer (guarantee that a message has got to the peer) and also guarantee the order of the delivery. Automatic routing protocols logically represented on this layer (they work on top of IP) are in a sense a part of the network layer protocols. For example, TCP provides the guaranteed delivery mechanism with a connection establishment prior to the communication providing an application with a reliable data flow and error-free delivery. Also it asks for a data resending in the case of data losses and removes duplicated packets. TCP controls a network load and provides in-order delivery [2].

UDP is a data transmission protocol to deliver datagrams without connection establishment [15]. Also it is called a protocol of unreliable delivery that means that there is no capability of assuring data delivery and packets may be lost or delivered not in order. In applications that require reliable data transmission TCP is used. UDP is usually used in applications like streaming video and computer games where packet losses can be tolerated and repeated requests are hard to do or it does not make sense. Also it is used in applications of a request-reply form (e.g. DNS requests) where the connection establishment takes more resources then the re-delivery. Both TCP and UDP use a number called a port number to define an upper layer protocol.

Application layer. Most of the network applications work on this layer. These applications have their protocol of data exchange like HTTP for WWW, FTP (file transfer), SMTP (e-mail), SSH (secure connection with remote machine), DNS (symbolic names to IP addresses conversion) and a lot more. Most of these protocols work on top of TCP or UDP and are associated with some port number. These port numbers are set by IANA (Internet Assigned Numbers Authority) [16].

The number of layers is the result of the following conditions that were taken into account:

1. A layer has to be created if a separated layer of abstraction is built.

2. Each layer has to perform a strictly defined task.

3. The choice of functions for each layer has to be done with respect to the creation of international standard protocols.

4. The borders between the layers have to defined so that the data flow through them is minimized.

5. The number of layers has to be sufficient so that different functions are not united without necessity. On the other hand, it has not to be too large so that the architecture is not cumbersome.

The Internet protocol stack describes system means of interactions. The applications implement their own protocols to interact. Some applications may implement the functions of the highest layers of the model and, in this case, they use only the underlying layer.

2.4 Summary

In this chapter, the main concepts of traditional data networks were discussed. When building a network one has to consider the following facts: what kind of channels to use, what is the network topology, what protocols to deploy in the network. To make this process easier, different protocol stack models were developed. In practice, the most used protocol stack model is the TCP/IP model developed by the IETF. Thus there is a structure to be followed when building a network.

The structure of the protocol stack model introduces protocol layers. A layer is a set of protocols that are used to solve some particular task. There are interfaces between layers that are used to use the functionality of lower layers by upper ones.

A client-server architecture was depicted. There are servers that contain most of the resource and clients that interact with the server to get the access to the resources. A server provide the access to its resource by implementing services.

3 HTTP

HTTP (Hypertext Transfer Protocol) is the protocol that is widely used in the Internet today. Its main usage area is web browsing. In challenged environments its functioning is limited. The main reason for this is that it requires several round-trips to fetch a single web page. To understand what kind of modification has to be done to HTTP to make it work in DTNs it is necessary to show how HTTP works in traditional TCP/IP networks.

In this chapter, the HTTP structure is described. Its main parts are analyzed. Request and response methods and headers are given. MIME is proposed as an additional protocol that can be used with HTTP in DTNs. MIME can be used to aggregate resources of a web page in a single object and to transfer them at once. The MIME structure and its main headers are given.

3.1 HTTP and the Web

HTTP nowadays is used everywhere in the World-wide web to get the information from web-sites. For example, in 2007 in North America the part of the HTTP traffic exceeded P2P networks that were leading for more than four years [17]. Nowadays almost half of HTTP traffic is a streaming video and audio. HTTP is an application layer protocol for data transfer (initially in the form of hypertext documents).

HTTP Features

The base of HTTP is a client-server technology so it is assumed that there are consumers (clients) that initiate connections and send requests and suppliers (servers) that wait for connections to get requests and produce necessary actions to return back messages with the results [18].

The main object of manipulation in HTTP is a resource which is pointed by a URI (Uniform Resource Identifier) in a client's request. Usually resources are files that are stored on the server but they can also be logical objects or something abstract. The feature of HTTP (the RESTful principle – REpresentational State Transfer) is a possibility to set in the request and in the reply the representation way of the same resource by different parameters: format, coding, language, etc. Due to this possibility a client and a server can exchange binary data through the text-based protocol [18].

The messages exchange goes according to the scheme "request-reply". For a resource identification HTTP uses global URIs. Unlike many other protocols HTTP does not maintain state. This means that the intermediate state between "request-reply" pairs is not saved. Components using HTTP may store the state information themselves. A browser sending requests may store the delays of the replies. A server may store the IP addresses and the headers of the last clients. But the protocol itself does not have a notion of previous requests and responses, it does not maintain state and these requirements were not considered in its development [18].

HTTP System Components

There are 3 conceptual categories of functions in HTTP:

- Servers as the main suppliers of the information storage and processing (request processing).
- Clients as the servers' services consumers.
- Proxies as the intermediate agents.

A proxy server is an intermediate agent that accepts clients' requests and depending on internal settings changes some parts or the whole request message or leave it as it is. Then it forwards this request to the server. At the moment of processing requests a proxy may work as a server and when sending requests – as a client. Additionally a proxy may support an internal cache with requests and responses. This cache stores server responses and returns them on client requests without sending those requests to the server. As a result the time of connection reduces and the performance of the working with remote or slow servers increases. But not all the responses can be cached. Some requests may contain parameters that restrict caching.

Proxies may have many functions. For example, there are *anonymous* proxies that are used to hide clients IP-addresses, *caching* proxies that speed up the access to resources, *content filters* that are used to restrict the access to some web services, etc.

To differentiate servers from proxies in the specification there is a term 'a server of origin'. Of course, the same software product may work as a client, a server or a proxy depending on the given tasks. In the HTTP specification the behavior for each of these roles is described.

Clients. Initially the protocol was developed to provide an access to hypertext documents in the world-wide web. That is why the main client implementation is a browser (a user agent). Popular browsers are: Chrome, Epiphany, Internet Explorer, Konqueror, Mozilla Firefox, Opera, Safari.

To view the saved sites content off-line there were developed offline browsers (HTTrack, Offline Explorer). If a connection is unstable download managers are used to get large files. They allow to continue the downloading process at any time after loosing a connection when the connection goes up again.

Virtual maps like Google Earth or NASA World Wind also use HTTP. Sometimes HTTP is used by applications to download new versions. A large set of robot-applications is used in searching systems [19].

Servers. The main implementations of the servers include Apache, Internet Information Services, lighttpd, nginx. *Proxy servers*: Squid, UserGate, Multiproxy, Naviscope.

HTTP Usages

Along with a simple fetching of documents there are more interactive uses of HTTP. One of these technologies is called AJAX. It allows to change HTML pages dynamically and communicate with a server in background [20]. An example of its usage is when a user fills in a form on a web page. When a user is entering data into the fields a browser exchanges with a server allowing a page to be changed. It may be used to propose the user possible alternatives for the field or to check the data immediatly and show the user where the problems are.

Although AJAX reduces the traffic between a client and a server it actully increases the number of "request-response" pairs and thus the number of round-trips. The number of round-trips is a critical value in DTNs. So the usage of AJAX in these environments is almost impossible.

There are other interactive technologies like Java applets [21], the Flash stack of technologies [22] and so on. They become more and more popular in recent time. And all of them are dedicated to create more interactive web pages, thus imposing problems for HTTP in DTNs.

3.2 HTTP Protocol Overview

HTTP allows obtaining an access to the resources of WWW servers. To unify the access to multi-functional network resources servers support a set of interfaces allowing to structure the levels and the methods of working with different network resources. URI, URL, URN are different names of (almost) the same object designated for the identification of types, working methods and nodes where some resources available through the Internet are located [23]. HTTP URI is comprised of three parts:

- *a protocol.* Identifies the type of service that can be used to get the access to the resource itself. E.g. http or ftp server.
- *an address.* Identifies a resource host, e.g. `www.example.com`.
- *a name or a path.* Identifies the whole path to the resource on the selected host that we would like to use to get access to the resource, e.g. `/path/to/image/image1.gif`.

HTTP can handle documents of any type. Documents on web-servers are usually presented in the HTML format. HTML stands for Hyper Text Markup Language and is used to provide information in WWW. Hyper Text or hyper-links make links inside a text document that allow a user to quickly go from one part of a document to another part or another document. A hyper-link contains a URL. By clicking a hyper-link to some page a user can move not only within one web node between its pages but also move to other web nodes with different resources that are located in some other places and probably use different methods.

There are two types of identifiers: an absolute URI and a relative URI. An absolute URI contains a whole object's address and a protocol. So it contains all the

above parts. The absolute URI `http://www.example.com/imgs/image.gif` corresponds to the file `image.gif`, located in the `/imgs` directory of `www.example.com`.

This means that the type of access is the access through HTTP. The protocol is separated with ":". Two slashes separate the address of `www.example.com`. Then there is the path to the file and the file itself.

An absolute URI always has to start with a protocol. If a path to a file and a filename are missing then it is assumed that the requested path is the root path of a server "/". A protocol and a DNS address of a server are case-insensitive, but a path and a filename generally are case-sensitive [23].

A relative URI uses the URI of the current document that was requested by an HTTP client. Using the same procedure a client reconstructs a URI changing only some of the filenames. If there is ".." before a relative URI then it means to go one level up in the directory tree. "/" as a starting symbol ignores all the directories and adds the given path to the server's address directly.

Also a resource URI may contain not only a resource name but some parameters that are required for its operation. A resource name is separated from the parameters string with "?". The string consists of assignments separated with "&". Each assignment consists of a parameter name and its value separated with "=". Symbols that are not ASCII are substituted with "%" and a hexadecimal value of a symbol. For example, "?" is changed to "%3F". Since a space symbol is quite often it is substituted not with "%20" but with "+". "+" itself is changed to "%2B". For a given resource the whole parameters string is a single parameter of a string type, so parameter type, order and uniqueness do not matter [23].

HTTP Paradigm

HTTP is based on the paradigm of requests and replies. A requesting application (that is called a client) establishes a connection with a serving application (a server) and sends a request to the server in a form: request method, URI, protocol version, that is followed by a MIME-type message, containing the control information of the request, the client's information and, maybe, a message body. The server replies with a message, containing a status line (that includes a protocol version and a status code – success or error) that is followed by a MIME-type message that includes the server's information, the meta data about the response content and probably the response body itself.

It has to be noticed that one application can be a client and a server simultaneously but mostly they are not. The usage of these terms in this context is related only to the role of an application during the given communication session but not to the general application functionality.

In the Internet, communications are usually based on the TCP/IP protocol. For WWW the default protocol is TCP and the default port number is 80, but other port numbers may also be used and it does not eliminate the possibility to use HTTP as a higher layer protocol.

For most of the applications a communication session is opened by the client for each request and is closed by the server after replying to the request. Nevertheless,

it is not a protocol feature. Both the client and the server have to have a possibility to close the session, for instance as a result of some user action. In any case, the session closure, that is initiated by any side, aborts the ongoing request regardless of its status [18].

Messages Structure

Each HTTP message consists of 3 parts which are transferred in the defined order:

1. A starting line defines a message type.
2. Headers characterize a message's body, transmission parameters and other information.
3. A message body contains message data. It is always separated from the headers with an empty line that contains only CRLF – carriage return, line feed pair.

Starting lines differ for requests and responses. A request starting line looks as follows:

```
Method URI HTTP/Version - for other versions
```

where `"Method"` is a request name, one word in capital letters. In the HTTP 0.9 only the `"GET"` method was used and there exists a list of methods for version 1.1. A URI defines the path to the requested document. `"Version"` is a pair of dot-separated digits, such as 1.0.

A starting line of a server's response has the following format:

```
HTTP/Version StatusCode ReasonPhrase
```

where `"Version"` is the same field as in the request. `"StatusCode"` consists of three digits. This code defines the following content of a message and the client's behavior. `"ReasonPhrase"` is a short text with the status code explanation for a user. It does not have any special meaning for servers and clients and is not compulsory.

Methods

The number of methods that could be implemented in HTTP was defined by two main principles: RESTful principle and CRUD – Create, Read, Update, Delete. The first principle requires that the protocol complexity and the number of methods is kept close to minimum. The second one tells what methods must be implemented. Each particular method is described later.

An HTTP method is a sequence of any characters except control characters or separators that points to the main operation on a resource. Usually a method is a short symbol literal written in capital letters. The method's name is case-sensitive. Each server should support the methods `"GET"` and `"HEAD"` at least. If a server did not recognize the method provided by a client then it has to return the 501 (Not Implemented) status code. If a server knows the method but it cannot be applied to

that particular resource then the 405 (Method Not Allowed) status code is returned. In both cases a server would include the "Allow" header with the list of accepted methods in the response message [18]. Apart from "GET" and "HEAD" methods the method "POST" is used quite often.

The "OPTIONS" method is used to determine a server's capabilities or communication parameters for a particular resource. In the response a server replies with the "Allow" header with the list of accepted methods. Also in the response headers the information about supported extensions may be included. It is assumed that the client's request may contain a message body where it is specified what information a client is interested in. The body format and the way of work with it is not specified at the moment. Servers ignore it [19].

The same situation is about the body in a server's response. To know all the server capabilities a client has to put an asterisk in a URI. The requests of a kind "OPTIONS * HTTP/1.1" may also be used (as an analog to "ping") to test the server support for the HTTP 1.1 version. The result of this method is not cached.

For example, to get all extensions supported by server.com a client sends the following message:

```
OPTIONS * HTTP/1.1
Host: server.com
Compliance: *
```

Then server.com responds as follows:

```
HTTP/1.1 200 OK
Date: Thu, 2 Sep 2010 15:34:12 GMT
Server: Apache/2.0
Public: OPTIONS, GET, HEAD, PUT, POST
Compliance: rfc=2616, rfc=1945
Content-Length: 0
```

The headers and the message body may be missing but the starting line is a compulsory element because it defines the request/reply type.

Request Structure

The first line of a request is as follows:

```
METHOD <SP> Request-URI <SP> HTTP-Version
```

<SP> is a space symbol. Each line is ended with <CRLF> – carriage return, line feed pair.

The "METHOD" field contains an identifier of a method that is used to access a server. A method defines the content and the structure of request headers, the type of parameters and their structure. The most common methods are "GET" and "POST". "HEAD", "DELETE" and "PUT" are also used sometimes. In the HTTP 1.1

specification some new methods were added, for example, "OPTIONS" and "TRACE". All methods identifiers are case-sensitive.

"Request-URI" contains an absolute or a relative URI. In case of an absolute URI the scheme is set to "http" and as an address there comes the address of the server which the client is connecting to.

"HTTP-Version" points to the version of HTTP that is supported by a client. The default version is 0.9. To set the version of HTTP the scheme "major.minor" is used. HTTP/1.0 is greater than HTTP/0.9. The example of the client request first line is:

```
GET /index.html HTTP/1.0
```

After the first line there are headers. Each header consists of the FieldName: FieldValue pair and ends up with <CRLF>. The order of the fields is not important. Header names are case-insensitive. Header values may be case-sensitive. The HTTP/1.0 specification requires that ":" follows the name immediately and between ":" and the value there must be only one space. HTTP/1.1 has less strict requirements. All headers may be divided into the following groups: General-Header, Request-Header, Response-Header, Entity-Header, Extension-Header. The most common headers for each group are:

- [General-Header] Date, Pragma, Cache-Control, Connection, Trailer, Transfer-Encoding, Upgrade, Via
- [Request-Header] Authorization,From,If-Modified-Since, Referrer,User-Agent, Accept, Accept-Charset, Accept-Encoding, Accept-Language, Expect, Host, If-Match, If-Range, If-Unmodified-Since, Max-Forwards, Proxy-Authorization, Range, TE, If-None-Match
- [Response-Header] Location, Server, Www-Authenticate, Proxy-Authenticate, Age, Retry-After, Vary, ETag, Warning
- [Entity-Header] Allow, Content-Encoding, Content-Length, Content-Location, Content-MD5, Last-Modified, Content-Type, Content-Language, Expires.

"General-Header" and "Entity-Header" group headers may be used both in client requests and in server responses. "Request-Header" fields are used only in client requests and "Response-Header" fields are used only on server responses. The "Extension-Header" group contains additional fields. A request head may look as follows:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Accept: image/gif, text/html, */*
From: webmaster@example.ru
If-Modified-Since: Mon, 22 Oct 2001 20:28:55 GMT
Connection: Keep-Alive
```

A head is separated from a message's body with an empty line. A message body may be contained in a client's requests when methods "PUT" or "POST" are used. If this is the case, it is necessary to give "Content-Length" header, too.

Request Methods

The "GET" method requests the information about a resource located at the given URI. Usually the requested information is an HTML file or some other text file or a graphic file, a video file or an audio file and so one. If an URI is associated with an executable file (e.g. a CGI service) a CGI process is run and a client receives the results of that was put to the standard output by the process.

A message's body when using "GET" is always empty. This is why all parameters that have to be passed to a server are attached to the URI in the first request line. For example, if a <form> tag of a document contains the method attribute value that is equal to "GET" then "key=value" pairs containing the form's data are attached to the URI after the question mark. The pairs are separated with "&". Non-ASCII symbols are encoded with their codes.

Also in the "GET" method the information about additional paths may be transmitted. In this case an additional path is given after the URL (after the filename and before the question mark), i.e. /cgi-bin/getacc.exe/mydoc.txt?login=noname. A server determines where a program name ends (getacc.exe). All the data following the program name and before the question mark are interpreted as an additional path (this path will be available in a CGI process via the PATH_INFO environment variable. A request with the "GET" method may look as follows:

```
GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Host: www.example.ru
Accept: image/gif,image/jpeg,text/html,*/*
```

In case of a data transmission to a CGI program:

```
GET /cgi-bin/getacc.exe?user=noname&pass=nopass&show=1 HTTP/1.0
Host: www.mail.ru
Connection: Keep-Alibe
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
```

The "POST" method is usually used to transfer data from a client to a server. The data have to be processed by a resource that is given in a URI. This method is useful when working with CGI services. The "POST" method transfers parameters to a URI resource in a message's body. So when using it there are no restrictions about the length of the parameters string.

If a <form> tag of a document contains the method attribute value that is equal to "POST" then "key = value" pairs with the data from the form are put to the message's body. The size of the message is given in the "Content-Length" header and this field is mandatory when using the "POST" method. Responses to the "POST" method are not cached. The example of the request is as follows:

```
POST /cgi-bin/getaccount.exe HTTP/1.1
```

```
Host: www.mail.ru
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Content-type: application/x-www-form-urlencoded
Content-Length: 32
/* blank line */
user = noname
&pass = nopass
&showdetail = 1
```

The "HEAD" method is almost the same as "GET" except that a client receives only the head of a response. This method is used when a client wants to get the data about a document but not the document itself. For this method there are a lot of applications. For example, a client may request the following information:

- *the time of a document's modification* (this data is useful for the requests related to cache)
- *the size of a document* (it is necessary to estimate the time of transmission and page rendering and to find out if it is possible to get a smaller version of the document)
- *the type of a document* (allows a client to get the documents of a specific type)
- *the type of a server* (allows to create specific requests)

The "DELETE" method is used to delete a resource. To process this method it is necessary for a user to have special rights on the remote URI. Usually this method is forbidden for unauthorized users. A "DELETE" method request may look as follows:

```
DELETE /pub/user1/temp.txt HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
```

The "PUT" method is used when a client wants to save a resource on a server. The resource itself is contained in the message's body. As well as for the "DELETE" method a user must have special rights for the directory where the resource is put. The size of a message's body is given in the "Content-Length" header. This field is mandatory when using the "PUT" method. An example of a "PUT" method request is:

```
PUT /pub/user1/upload/newpage.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Content-type: text/html
Content-Length: 60
```

```
<HTML>
  <BODY> My new page is under construction
  </BODY>
</HTML>
```

In the HTTP/1.1 specification some new methods were added.

The "OPTIONS" method requests the information about the communication parameters of a server. The requested resource is not downloaded and if it is a CGI module it is not executed. To request the information about the whole server '*' is used as a URI of the request. The "OPTIONS" method request may look as follows:

```
OPTIONS * HTTP/1.1
Host: www.example.com
User-Agent: Mozilla /4.0 (compatible; MSIE 5.5; Windows 98)
```

The "CONNECT" method is used by proxy servers that support SSL tunneling.

Response Structure

The first line of a response is

```
HTTP-Version <SP> Ret-Code <SP> Comment-String
```

The version field contains an HTTP version that is used by the given server to this response. "Ret-Code" is a 3-digits number that shows the request processing result. The format of the code is like in other application protocols of the TCP/IP stack like FTP and SMTP. The first digit defines the code class and each class contains a group of status code values. There are five possible values of the first digit:

- *1xx: informational codes* – the request is received, the processing continues;
- *2xx: success codes* – data were received successfully, they were processed and used;
- *3xx: redirection codes* – to process the request some additional actions are required;
- *4xx: client error codes* – the request contains syntax errors or requested resources are not available;
- *5xx: server error codes* – the server was not able to process and accomplish the request.

"Comment-String" is a description that simply is a text understandable by a user. It explains the returned value. The most common return codes are given in Table 1.

The status code `100 Continue` tells a client that the request part that is received by a server is accepted and the client should continue sending the remaining part. `200 OK` means that a request was processed successfully. `201 Created` is returned

Code	Reason	Code	Reason
100	continue	400	bad request
200	OK	401	unauthorized
201	created	403	forbidden
202	accepted	404	not found
204	no content	405	method not allowed
206	partial content	407	proxy authentication required
300	multiple choices	408	request timeout
301	moved permanently	413	request entity too large
302	moved temporarily	414	request-URI too long
304	not modified	500	internal server error
305	use proxy	501	not implemented

Table 1: HTTP Status Codes

when a new resource was created when processing a request. Headers point to the new resource location. `202 Accepted` is used when a request is processed but the processing is not completed. `204 No Content` informs a client that a request was processed successfully but there is no body to return as a result of it. `206 Partial Content` is used when a client has requested some part of a document.

`300 Multiple Choices` return code tells a client that there are several representations of the same resource and one of them should be chosen. `301 Moved Permanently` indicates that the requested resource URI is changed and returned URIs should be used in future references to this resource. The return code `302 Found` means that the requested resource URI is changed but in future it may be changed back so the old URI should still be used. `304 Not Modified` is returned when a client issued a conditional request and the document was not modified since the date given in the request. `305 Use Proxy` forces a client to use a proxy to get the document.

`400 Bad Request` is sent when a server was unable to parse the request. `401 Unauthorized` shows that a user authentication is required to proceed. The status code `403 Forbidden` is used when the request will not be processed due to some reason that should be explained by the server. `404 Not Found` is used when there is no resource available or a server hides the actual reason for rejecting the request. `405 Method Not Allowed` tells a client that the requested method cannot be used with the given URI. `407 Proxy Authentication Required` is similar to `401 Unauthorized` but the authentication is required by a proxy. `408 Request Timeout` means that a client did not issue the request within the time a server was waiting for it. `413 Request Entity Too Large` allows a server to discard processing of too big requests. `414 Request-URI Too Long` is returned when a server rejects to proceed the request because URI is longer than the server is able to handle.

`500 Internal Server Error` indicates that a server ran into the situation that prevents it from proceeding the request. `501 Not Implemented` is returned when

a server does not support the necessary functionality. `505 HTTP Version Not Supported` is used when a server does not support the client's HTTP version.[19]

For example, the first line of a server's response may look as follows:

```
HTTP/1.0 200 OK
```

or in case of not finding the requested document:

```
HTTP/1.0 404 Not Found
```

Then follows a head which format is the same as a request head. The head is separated from a message body with an empty line. The size of the message body is put to the `"Content-Length"` header. It is allowed not to include the `"Content-Length"` header in a response. In this case a client may determine the size of the message by the moment of closing the connection.

Also in the HTTP/1.1 specification with the help of `"Transfer-Encoding"` it is possible to define the "chunked" type of transfer. In this case the size of the message's body is defined in the body itself and the `"Content-Length"` field may also be missing. So a response head may look as follows:

```
Date: Thu, 09 Feb 1997 07:25:33 GMT
Server: Apache/1.2.6
Last-modified: Sun, 11 Jun 1997 20:49:05 GMT
Content-type: text/html
Content-length: 3967
```

3.3 Request and Response Headers

General headers may be used both in requests and responses. `"Date:"` contains the date and time of the message construction. The format is described in RFC822 [24]. `"Pragma:"` is used to set special directives. For instance, the `"no-cache"` option does not allow intermediate nodes to use the cache when constructing response.

The `"General-Header"` group in the HTTP/1.1 specification was extended with the following fields:

`"Cache-Control:"` allows to control caching in more flexible way. It allows to set time and object caching restrictions, to control cache requests from some clients.

`"Connection:"` controls connection parameters. In HTTP/1.0 a client has to provide the `"Keep-Alive"` value in this field to have a permanent connection. In the HTTP/1.1 specification permanent connections are used by default so there is no need to set the `"Keep-Alive"` value for a HTTP/1.1 client.

If a client does not want to set a permanent connection then it sets `"Close"` in the `"Connection"` field. In this case a server will close a connection immediately after sending a response. In the HTTP/1.1 specification the `"Close"` value may also be given in a response. By doing it a server informs a client that the connection will be closed immediately after transmitting a response and a client must not send requests to this connection anymore.

At the receiving of a "Keep-Alive" value, a server may respond with another "Keep-Alive" value and include some parameters of a permanent connection. For example, a timeout value after which a connection is closed if no data come from a client.

"Transfer-Encoding:" is used to define a coding type of a message's body or a response when transmitting it over a network. In the HTTP/1.1 specification the most used type is "chunked". In this case a body is transferred in chunks. For each part in the first line there goes the size of the chunk. Then comes the chunk itself ending up with "<CRLF>". The last chunk has zero size and may contain additional parameters. So a message of any size may be transmitted without setting the "Content-Length" field. It can be useful when generating HTTP-responses dynamically.

"Upgrade:" contains other protocol specifications that can be used to set a connection to a server or a client except the current protocol. For example, a client may advertise its desire to use a higher version of HTTP even though the current request is made using the lower version. This field is introduced to make the transition to newer protocols easier. "Via:" is used to log intermediate proxy agents or gateways that were passed by a request or a response.

For example, a "General-Header" head may look as follows:

```
Date: Mon, 18 Nov 1998 07:34:28 GMT
Pragma: no-cache
Connection: Close
```

The "Request-Header" group headers can only appear in requests. It allows clients to send additional information to servers.

"Authorization:" contains the information needed for a user authentication and determining his rights. Responses to the requests with the "Authorization:" field are not cached by intermediate nodes. The "From:" field may contain an e-mail address of a user. The field may be used to collect statistics about the web-site attendance.

"If-Modified-Since:" is used along with the "GET" method. If a resource was not changed since the moment given in this field then the resource is not returned. As a response comes only the head and the response code. This field may be used to control cache contents.

"Referrer:" contains a URI of the previous resource. This parameter may be used by a server, for example, to optimize caching or to build search paths, etc.

"User-Agent:" contains the information about a client's software.

In the HTTP/1.1 specification the "Request-Header" group was extended.

"Accept:" asks a server to use only specified data formats in the response. For example, if a client understands only text data and "gif" images the request has to contain:

```
Accept: text/*, image/gif
```

"Accept-Charset:", "Accept-Encoding:" and "Accept-Language:" define acceptable symbol charsets, coding types and languages. Also a request may contain

a "Content-Language:" field that describes national languages that are used when constructing a response body.

"Host:" contains an IP or a DNS address of the host that contains the requested URI. Generally after ":" an HTTP port that is used by this host may be given. If there is no port then by default the port number 80 is used. This field can be useful for transparent proxies. Usually when using the "Host" field in a request a relative URI is given. So it does not contain the host address but this address is given in the "Host" field. A request may pass through a chain of proxies without setting any additional options and fields. This field is mandatory when using HTTP/1.1.

```
GET /examples/mainpage.html HTTP/1.1
...
Host: www.example.edu:8000
```

"If-Match:", "If-None-Match:", "If-Range:", "If-Unmodified-Since:" set different flags on the previous requests of the resource to control server cache more efficiently.

"Max-Forwards:" contains the maximum number of intermediate nodes that can be passed by a request. This field may be used only with "TRACE" and "OPTIONS" requests. Each intermediate agent reduces this number by 1 before re-transmitting the message to the next node or the final server. If upon the receiving of a request this field contains zero then an agent should not forward the request further but it should construct a response for the client.

"Proxy-Authorization:" field allows a client to register on hosts with intermediate agents. This field may serve to authenticate the client on the whole way to the requested resource.

"Range:" allows to control the size of a message. With the help if this field it is possible to get not the whole document but only a part of it. This allows receiving the resource in parts or to get only the remaining part of the resource when the connection was broken. As a parameter of this field there goes units in which parts are measured and then after the "=" sign there should be given one or more ranges, separated with ",". For example, "bytes=0-19,100-149,-10" allows to get the first 20 bytes, the 50 bytes starting from 100 and the last 10 bytes.

A "Request-Header" head may look as follows:

```
Host: www.example.ru
Authorization: Base B2Uda29091N1ploasq==
If-Modified-Since: Wed, 1 Sep 1999 20:34:28 GMT
Referer: http://www.example.ru/home.html
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: ru
Accept-Encoding: gzip, deflate
Range: bytes = 0-99
```

The "Response-Header" group allows the transmission of additional information about request processing that cannot be put into a status line.

The "Location:" field contains an absolute URI of the created or moved resource. For instance, if a client sent a request to a server to get some document that was moved to another directory or another server and the server responds with the "Location:..." field, then the client's request is redirected to the given URI automatically.

"Server:" contains the specification of the software that is installed on the server.

"WWW-Authenticate:" contains the parameters of an authentication scheme and realm. In HTTP/1.1 the "Response-Header" group was expanded.

"Age:" contains the number of seconds a server spent when sending a response to the given request. The field allows to control productivity of different servers and intermediate nodes.

"ETag:" contains key words of the resource. According to them the resource is identified on a server and in caches of intermediate agents.

"Proxy-Authenticate:" contains parameters of the scheme and space of authentication for a proxy server.

"Retry-After:" tells a client to retry the request after the given number of seconds and after the given date.

"Warning:" contains additional information about the status of the request processing. This is a kind of a system with additional response codes. An example of a "Response-Header" head is:

```
Location: http://www.example.ru/newsite/newhome.html
WWW-Authenticate: basic realm="MY Securit"y
Server: Apache/1.2.6
```

The "Entity-Header" group contains the information about the structure and the format of a message's body. If the body is missing then it contains the information about the request resource. It can appear both in requests and responses.

"Allow:" contains the list of methods that are supported by a resource. This field has an informational meaning only.

"Content-Encoding:" contains an additional encoding type of the resource (to use the resource it has to be decoded by the given algorithm).

"Content-Length:" defines the length of a message.

"Content-Type:" contains the type of a resource and the data encoding (us-ascii, iso8580-1, windows-1251, koi8-r, etc. according to the MIME specification).

"Expires:" contains the resource expiring date. This field may be used in caching mechanisms as a parameter to forbid caching of old resources. This field also may inform a user about the fact that the information of this resource is too old.

"Last-Modified:" contains the date and time of the last modification of a resource.

In the HTTP/1.1 specification this group was expanded. The fields of the "Content*:" family ("Content-Base", "Content-Range", "Content-Location",

"Content-MD5", "Content-Language") describe respectively a format, parts sizes, a location of resource parts (if there are parts), a cryptographic algorithm, a national alphabet.

An "Entity-Header" head may look as follows:

```
Allow: GET, HEAD
Content-Encoding: x-gzip
Content-Length: 4597
Content-Type: text/html; charset=windows-1251
Expires: Mon, 01 Sep 2000 14:01:00 GMT
Last-Modified: Sun, 03 Sep 1999 10:08:22 GMT
```

The "Extension-Header" group contains fields that can be added to the specification later. All the fields that a client cannot recognize should be ignored by a client. Proxies should transmit them without changing. So the functionality of HTTP can be expanded without changing the whole structure of requests and responses. Some of the fields from the HTTP/1.1 specification are added to the HTTP/1.0 specification, too. This is done to provide compatibility. On of these fields is "Connection:" that was missing in the HTTP/1.0 specification.

In the HTTP/1.1 specification some of the fields were added to work with proxies. For example, "X-Forwarder-For" is used by proxies to set a client's IP address. The "Proxy-Connection" field controls parameters of the connection with a proxy.

HTML Headers

The HTML markup language allows the setting of necessary values for HTTP headers inside the "<HEAD>" part of a document with the "<META>" tag. A header's name is given in the "http-equiv" attribute, and a value is put to the "content" attribute. Almost always the "Content-Type:" header is set to avoid the problems with a browser's representation of a text. Also setting the "Content-Language:" header value is a good practice [19].

```
<meta http-equiv="Content-Type"
content="text/html;charset=windows-1251">
<meta http-equiv="Content-Language" content="ru">
```

Real Request/Response Example

The following is the example of a real request that is sent by the client – Mozilla Firefox browser:

```
GET / HTTP/1.1
Host: sysman.ru
User-Agent: Mozilla/5.0 (Windows NT 5.1; ru; rv:1.8.1.2)
Gecko/20070219 Firefox/2.0.0.2
Accept:
```

```
  text/xml,application/xml,application/xhtml+xml,text/html;
  q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: ru-ru,ru;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1251,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

At the first line of the request the first word represents a method by which data is sent. Then follows the request until the script regarding to the server's root. Then follows the protocol to be used during the transmission (HTTP/1.1). The second line determines the name of the server from where the data is requested. As a third line the browser identifies itself, sending the browser type, the operation system version, the default language and the browser version.

In the next lines the browser sets what types of documents it can accept, the preferable language, available compression types during the transmission and also the preferable encoding. The pre-last line points to the amount of time during which a connection should be kept alive. And the last line shows that this connection should be kept alive. In principle, our browser sends a lot of unused information and the request can be shortened to:

```
GET / HTTP/1.1
Host: sysman.ru
Connection: close
```

The response of the server to the request is:

```
HTTP/1.x 200 OK
Date: Thu, 17 May 2007 14:07:42 GMT
Server: Apache/2.0.59 (FreeBSD) PHP/4.4.6
Set-Cookie: session\_id=4f5df19ee96008635c02eac3dcbc1354; path=/;
domain=.sysman.ru
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=WINDOWS-1251
Content-Language: ru
```

At the first line of the response the first word points to the protocol that is used during the transmission (HTTP/1.x), the status code and the status description. At the second line the GMT time and date on the server is sent. At the third line the server describes the software used. Then the server sets a cookie.

3.4 Authentication Mechanisms

HTTP provides an easy mechanism for authentication of server's users. The mechanism is based on the exchange of authentication information between a client and

a server that contains a resource. The mechanism allows to choose the authentication scheme and the level of secrecy of the data transmitted. In the following the most popular scheme that is used for authentication ("Basic") is described. A client requests a resource that has a restricted access.

```
GET /secret/security.html HTTP/1.0
```

In this case the server responds with the 401 status code that means that the authentication is required and sets the `"WWW-Authenticate:"` header. It contains the scheme and at least one realm name.

```
HTTP/1.0 401 Unauthorized
Date: Monday, 02-Feb-00 22:02:11 GMT
Server: HERBER 4.3
MIME-version: 1.0
WWW-Authenticate: basic realm="Herber Security"
```

The whole space of server's resources may consist of several authentication realms. Each realm may use its own authentication scheme. The `"Realm"` parameter contains a symbol link. Along with a URI this link defines the resource uniquely in a network. In the example the realm is "Herber Security" and the scheme is "Basic".

This scheme assumes that a client has to pass its identifier and the password for the realm given. If a server has several realms and each realm requires its own parameters for authentication (user id and password) then to use any of the resources of the realms a user has to register in each realm separately.

After receiving the server response with an authentication request a client has to send the request again and include the `"Authorization:"` header with the string of a user id and password in base64 encoding – this is the only encoding type that is used with the "Basic" scheme. Usually a browser shows a dialog to ask a user her id and password. Then a browser constructs the string "id:pass", encodes it with the base64 algorithm and sets the `"Authorization:"` header.

After processing this request a server has to allow a client to use the resource. If the authentication failed then a server returns the 403 code (forbidden). The "Basic" scheme does not assume any encrypting of the user data because the base64 algorithm is designated for encoding but not for encrypting. So this scheme is mostly used for tests, identification and to log the users activity rather than for protection servers resources.

Additionally, HTTP provides a challenge-response authentication mechanism that was developed to avoid the most serious flaws of the basic scheme. It is called the Digest Access Authentication scheme [25]. When a client tries to access a restricted resource the server sends back a challenge that contains the realm, the nonce value, the algorithm to be used in the response and some other parameters. The client then sends back a checksum of the username, the password, the nonce value, the HTTP method and the requested URI. So the password is not transmitted in clear text over the net.

Both of these scheme are weak. And although the “Digest” scheme is better than the “Basic” scheme it is still very easy to eavesdrop most of the message. The “Digest” scheme actually protects only the user password so that it is not sent in clear text over the physical network. The request and the response, including the document itself, can be eavesdropped. It leads to the necessity of use of TLS (SSL) security whenever a reliable and closed data channel is required in the transaction [26].

3.5 HTTP Characteristics and Limitations

As the main characteristics of HTTP there are the following: simplicity, extensibility, wide deployment.

Simplicity. The protocol is so easy in implementation that it allows to create client applications easily.

Extensibility. It is possible to extend the protocol features with the means of adding new headers still having the compatibility with other clients and servers. They will ignore unknown headers whereas it is possible to get a new functionality while solving a specific task.

Wide deployment. When choosing the HTTP protocol to solve particular tasks the important fact is its wide usage. As a consequence there is a lot of protocol documentation in different languages, included tools for the HTTP development in popular IDEs, the protocol is supported as a client by a lot of programs and there is a wide choice among hosting companies with HTTP servers.

Stateless approach. The architecture style that is used in HTTP assumes that there is not state that should be kept in servers, clients and proxies. It leads to the fact that it is possible to close a connection after the first “request-reply” pair and open another connection if necessary to get some other parts of a document.

Also this protocol has several limitations.

Big message sizes. The usage of text-based messages in the protocol leads to big messages compared to binary data. Because of that the devices load when creating, producing and transferring messages is increased. To solve this problem in the protocol there are included tools to cache data on the client side and also the possibilities of a content compressing during the transmission. By the specification documents of the protocol there may be included proxy servers that allow a client to get a document from the closest server. Also in the protocol there was introduced delta-coding for a client to get not the whole document but only a part that has changed [19]. However it should be noticed that message body sizes increase. So the content of a message is even bigger than the headers and this limitation becomes less important.

No navigation. Although the protocol has been developed as a tool to work with server resources there is no notion of navigation through the resources in a clear form. For instance, a client cannot request the list of accessible files like in FTP. It was assumed that a user knows a URI of the desired document. When it is downloaded the navigation would be done according to hyper-links. It is acceptable for a human-being but creates difficulties when the task includes an automatic processing and

an analysis of all the server resources without human participating. The solution of this task should be done completely by application developers that use this protocol. Though mostly it is the main idea behind HTTP that the navigation is done via hyper-links and the only resources that are accessible are those that are referenced to. So this is a design question rather than a protocol disadvantage.

3.6 Summary

This chapter has analyzed HTTP to find out share parts that have to be modified for delayed environments. HTTP is a powerful protocol. It can be easily extended and this is a good property for its modification. On the other hand, however, it cannot be easily deployed in challenged environments due to the fact that originally it was developed mainly to work with TCP and TCP assumes short delays. The second reason is the one-by-one fetching of the resources. In networks with long delays downloading a single web-page that refers to several external resources with traditional HTTP is almost impossible.

As a companion protocol that can be used to modify HTTP for networks with great delays there is MIME. Initially developed to transmit different kind of resources in e-mail systems it enables transmitting web pages as a whole too.

In the next chapter, the main protocol of DTN is described. HTTP modifications are introduced. The theoretical background of this chapter allows proposing these modifications and, finally, to build a DTN-enabled web browser.

4 Delay-Tolerant Networks

New approaches to networking are required when it comes to challenged environments. Long delays and link failures make it impossible to use traditional protocols in them. New protocols have to be developed. One of these protocols is the Bundle Protocol (BP) – the main protocol of DTNs [27].

In this chapter, the structure of DTNs is described. They are compared to traditional telecommunication networks. The main problems of working in delay-tolerant environments are stated. Reference solutions are explained. The BP is introduced. Finally, the HTTP modification that allows providing web access in DTNs is described.

4.1 Data Transmission Problems

Research work focuses on the creation of mobile networks that will be able to support communications even in the conditions of link failures and long delays.

The development directed to the construction of DTNs is stimulated by the needs of military, researching is dedicated to emergency service wireless networks that usually suffers from the lack of stability and predictability that are common for traditional wired communication systems. The DNS service and routers containing the most relevant information about other routers and their IP addresses will provide a browser with a requested web-page in a few seconds in the worst of cases.

But in networks dealing with disruptions this infrastructure either does not work or is not available at all. To have a possibility to deal with such situations researchers have modified routing principles. In DTN, messages can be sent by an initiating node even in the case when an IP-address of a destination node is not known due to disruptions that breaks a communication with name servers and routers.

Network usage scenarios are based on many assumptions. In case they do not hold, the functioning of a network becomes problematic or even impossible. They include:

– an end-to-end path between a source and a destination exists;

– delay times on the path are not significant;

– the probability of a packet loss is small [28].

Before sending any packets a node must resolve a domain name into an IP-address and this IP-address into a MAC-address. These procedures require the infrastructure that includes DNS servers, routers, gateways, etc. When there is no such infrastructure the work of the network is broken on the IP layer.

To establish a TCP connection a 3-way handshake has to occur, thus the time of connection-establishing is proportional to the round-trip delay [29]. Obviously the TCP/IP scheme is not acceptable in networks with very lengthy delay times [30]. Additionally high latencies drastically reduce TCP throughput performance. And frequent packet losses cause TCP to keep retransmitting packets until it decides that there is no path anymore and drops the connection completely.

When HTTP comes to delayed environments the situation becomes worse. The main problem is that usually HTTP requires several round-trips to get a single web page because for each embedded object it issues a new request and thus the number of request-response pairs grows with the number of objects that the web page contains. The worst case scenario is the usage of some interactive technology, like AJAX.

Most of the Internet protocols problems in challenged environments come from the idea of fate sharing [28]. It means that the state of a connections should be kept only in end nodes because when one of them fails it makes the connection useless. This leads to the fact that the end-to-end path should exist and since the reliability is based on retransmissions it also suggests that the delay times along the path are short.

So when some of the above assumptions do not hold the Internet protocol stack fails to do its task. There were some efforts to create an overlay on top of it that consists of proxies to deal with link drops and lengthy one way delays. But the use of proxies is a fragile approach because its configuration depends highly on the particular environment and do not provide a common protocol to interconnect challenged networks [28]. Thus a new architecture had to be developed.

To solve TCP/IP's problems in DTNs, a new protocol is developed and implemented. This protocol is called a DTN protocol and is based on a paradigm: to store and to forward. The DTN protocol (BP) works with bundles. A bundle is a message that contains both an application payload and routing-related information. Nodes of a DTN network exchange bundles and store them when there is no link to the next hop. When this link becomes available a bundle is sent towards its destination. This process is repeated until the destination is reached or the bundle's lifetime is expired.

Initially, DTN was designed for deep-space communications [31]. DTN started out as an interplanetary Internet. DTN differs from the traditional Internet protocols (TCP or UDP). Its feature is data delivery independently of the current links state. The classical protocols in a case of no possibility to deliver data immediately delete the data. In DTN the other principle is used: store and forward. When data for a node that is not available now are received they are stored. After finding a route to that node (or a destination node itself) the data is transmitted to the next node.

IP is a packet-based store-and-forward protocol, too. The difference of DTN from IP is in message sizes and their lifetimes. An IP-packet is discarded in a router immediately when its queue is full or there is no route to the next hop. The size of the packet is limited. The bundle size is theoretically unlimited and its lifetime may be much longer than of an IP packet. The bundle protocol principles are similar to mail principles. DTN nodes exchange bundles storing them when necessary in the same way as a letter travels from the sender to the destination through different post offices.

The classical protocols can be compared to a goods transportation in a truck from point A to point Z in a direction Node 1->Node 2->...->Node N. If at some point M there is no direction to point N, or the road gets lost, then the goods (data)

are discarded. The DTN protocols are able to transfer the information from node to node as the links restore and store the information in memory devices until the link is up again.

The basic version of the routing protocol for DTN has been developed.

4.2 DTN Architecture and Services

The DTN architecture has developed from the Interplanetary Internet that was an effort to bring the Internet communication style to the deep space challenged environment. In general the DTN architecture is a way of interconnecting heterogeneous networks. DTN is an overlay on top of these networks [32]. It provides an end-to-end message-oriented layer that exists above the transport layer and below the applications. This bundle layer is used to deal with underlying network disruptions. It hosts a persistent storage for messages that are waiting for a contact to appear in order to be forwarded.

The BP is accepted as a main transport protocol in DTNs. The architecture of a DTN is layered similarly to OSI. It means that the BP has two interfaces: it provides its capabilities to the upper layer (applications), and it uses a so-called convergence layer to actually perform the transportation tasks. There are several convergence layers available. They include TCP and UDP, and SCTP support is about to appear.

A convergence layer is the one that takes care of actual transmission. It deals with the underlying network properties. Thus the BP is independent of the underlying network. In the BP there are no timers and it is one of the main conditions to work in DTNs.

Generally, bundles present themselves as containers for application payload. They include the routing information but the routing itself is left to a convergence layer. Reliability mechanisms are implemented in the convergence layer, too.

A DTN-enabled application sends messages of an arbitrary length. These messages are called Application Data Units [32]. They are transfered in bundles. A bundle also contains the information related to routing that includes source and destination names. DTNs support a flexible naming scheme that is based on URIs. A DTN node is associated with one or many Endpoint Identifiers (EID). An application that wishes to accept bundles registers itself within a node. It is similar to binding to a socket.

DTNs support a feature called “late binding”. It means that the interpretation of EID need not happen in the source node. This is how DNS problems are solved. In challenged environments late binding may be beneficial because the time of message life may be longer than the validity of a binding so the binding at the source cannot be done. Also it reduces the amount of administrative information in a network [32].

Bundles may be of different service classes. There are 3 service classes defined: bulk, normal and expedited. They define the relative priority of bundles. It means that normal bundles are sent prior to bulk bundles, and expedited bundles are forwarded prior to both bulk and normal bundles [32]. In DTNs the quality of service cannot be guaranteed since it is unknown how long does it take to deliver a message.

These priority classes are applied only within one node, so a bulk bundle from one node may be forwarded before a normal bundle from another node depending on the forwarding policy.

Along with ADU and a routing information, a bundle also contains the priority information, a timestamp and some useful lifetime value, and some options. Most of the options are related to the reports, for example about bundle reception, forwarding, deletion. A set of options is used for so called custody transfers. This feature allows to transfer the responsibility for reliable delivery of a bundle from one DTN node another [32]. When Custody Transfer Requested option is specified it means that the next node upon receiving the bundle stores it and takes care of its further forwarding. This node is called a custodian. So instead of end-to-end reliability a custody transfer offers an option for hop-by-hop reliability.

At the bundle layer, the DTN architecture provides a framework for routing and forwarding. Generally this task is about finding a path on a multigraph where vertices may be connected with more than one edge. Additionally, the edges vary in time because some links go up and down periodically. When a link capacity is strictly positive during some period of time it is called "a contact". So a contact is a time period between two consecutive link drops. Data transmissions happen during these contacts. If the times of contacts are not known in advance the problem of routing computation becomes especially challenging and it is an active research area [32].

Since the duration of a contact may be small it leads to the possibility for a bundle to be fragmented. There are two types of fragmentation capabilities included in the DTN architecture: proactive and reactive. Proactive fragmentation means that a sender splits data into small pieces and sends them as separate bundles. It is up to the destination application to reassemble them. Reactive fragmentation happens when a link goes down during the transmission process. If some part of a bundle was reliably delivered then it may become a new bundle at the next hop and the rest of the original bundle would be sent as a new bundle by the sender when the link appears again [32]. This feature is not required to be implemented in every DTN node because it requires some functionality from the underlying protocols that may not be present and imposes troubles for authentication.

The nodes of a DTN are expected to be synchronized in time by some other means. The notion of time is needed for bundle and application expiration computation. It is also used in routing when the time of a contact can be predicted or is known in advance. Finally, the time synchronization is used for bundle and fragment identification [32].

When the DTN architecture was developed security considerations were taken into account because resources in DTNs may be very limited. The resources in DTNs are storage and capacities. One important requirement is that unauthorized traffic should be discarded as soon as possible because it may waste a lot of resources when being transmitted over a long distance. In the DTN architecture a security model is standardized but it is optional. This is done because many modern authentication systems were developed in low-delay environments thus they rely on the existence of some infrastructure. For example it may be assumed that a server to check

certificates is available. In challenged environments it may not be the case. In the DTN architecture there are two types of security schemes: hop-by-hop and end-to-end. The end-to-end authentication is used to ensure that an ADU was originated by an authorized user. The hop-by-hop security is used to determine if a node is eligible to communicate with another node, i.e. the next hop [32].

The bundle protocol works on top of different transport protocols and not all of them provide the same functionality. Hence there are so called convergence layers that are used to augment and adapt the protocol above and beyond the bundle protocol for its purposes. For example, there are TCP and UDP convergence layers. The complexity of a convergence layer depends on the underlying protocol. For example, if there is no reliable delivery provided by it then it should be implemented within the convergence layer whereas TCP convergence layer is rather simple.

The main features of the BP are:

- Custody transfer – the ability for a bundle agent to take full responsibility for a bundle reaching its final destination.
- Ability for implementations to cope with intermittent connectivity if required.
- Ability for implementations to cope with long propagation delays if required.
- Ability to take advantage of scheduled, predicted, and opportunistic connectivity (in addition to continuous connectivity).
- Late binding of endpoint identifiers in the overlay bundle network to network addresses in the underlying constituent networks.[8]

Sample Application

One example of a DTN-enabled application is a DTN-based blogging [33]. It was implemented as a special assignment in the networking laboratory of Aalto.

Traditional blogs are usually hosted on a web server along with a database that contains the data. It is based on the HTTP protocol to create new blog entries when an author uses a web browser to connect to the server. Then, through the web interface, a new entry is created and saved. If there is no TCP connection, for example if a user tries to create an entry from her mobile phone, then this scheme does not work [33].

In the application for DTN-based blogging the scheme is the same except for the fact that HTTP is not used when entries are created. Instead the new posts data are packed into bundles and is sent over a DTN network to the server. The connectivity is not required and the bundles are sent whenever the next hop becomes available [33].

4.3 HTTP over TCP and DTN

Similarly to TCP/IP breakdown in delayed networks HTTP introduces additional challenges. The scheme of HTTP work is based on the fact that a browser receives

some portion of data in response to its initial `GET` request. Usually this data is an HTML page. The page may contain embedded objects (e.g. pictures). To get them additional requests are initiated by a browser. This interactive exchange of messages is impossible in DTN networks. So HTTP has to be partly changed to work in DTNs. The modification is proposed in [1]. To get rid of the interactive messages exchange a DTN-enabled web server sends not only the requested web page but also all the embedded objects in the response to the initial `GET` request.

As it was discussed earlier, HTTP was developed to work in networks where delay times are not significant. If the delay time is great, a server does not respond and the 404 error – 'Resource is not found' is displayed[19]. For delay-tolerant networks this protocol, therefore, has to be changed.

To understand the problem statement and its solution methods, it is necessary to have a look at the term "bundle" in more details. When using HTTP in traditional networks, a browser communicates with a server according to the following scheme: the browser sends a request to the server, then the server sends to the browser the requested web-page that may include references to some resources embedded into this page.

The server does not send these resources immediately. To get them the browser issues new requests to the server, one by one for each resource. This process repeats until the last resource from the page is sent to the browser. Obviously in DTN networks this structure fails. Since the server has to send the resources one by one, it significantly increases delays. The page loading time grows and when there are periodic link breaks and the delay is huge, the browser may stop waiting for all the resources. Thus the browser will not be able to load the whole page. This traditional HTTP over TCP scheme is shown in Figure 1.

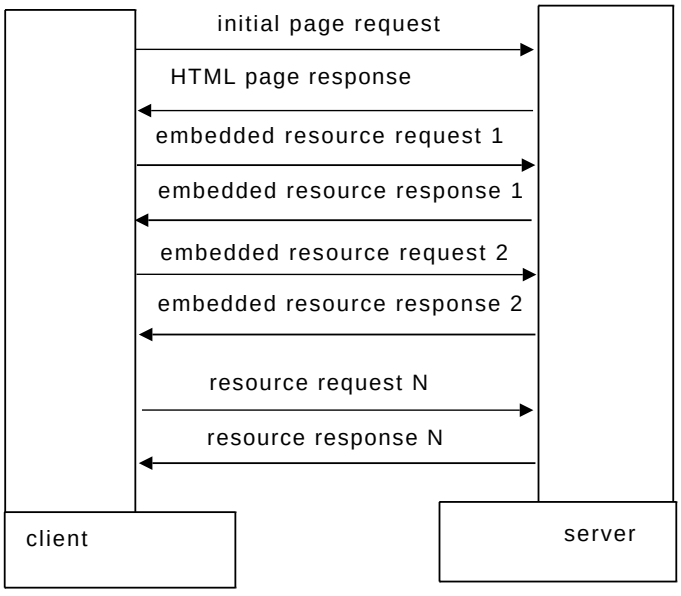

Figure 1: Traditional HTTP over TCP scheme.

The Bundle Protocol

The main idea of the method that is used to solve the given task is to modify HTTP to support the BP. The easiest solution of this problem is to map HTTP messages to bundles one to one. This method works fine for simple request messages but for retrieving multiple resources it is not so good because it will result in multiple round-trips. As we noted before it is necessary to aggregate multiple resources into one response in the form of a bundle to reduce HTTP's chattiness.

When a server receives a request it collects all resources that depend on this request, puts them into one bundle and sends the bundle back. So with the help of resources bundling it is possible to reduce the number of HTTP transactions from several to one. The number of resources on real Internet pages are usually far more than one. That means that this approach will be good in the networks tolerant to delays. To carry a lot of resources in one response bundle it is necessary to build the aggregation format for the resources. The next statements describe what requirements the format has:

1. Provide the way to point to the URI of the embedded resources.
2. Provide necessary metadata.
3. Do not introduce a significant overhead.

The first requirement is compulsory: a client has to know that these resources were sent according to the content of a web-page.

The second requirement refers to the specification of HTTP resource headers which is sent by the server. For example, the HTTP `"Expires"` header tells when the given resources is considered outdated. Until the date given in this header a local storage pool may service the resources since new versions of them are not required to be retrieved from the server.

Another example is the HTTP `"Content-Encoding"` header. Without it a client does not know how to interpret the file and may put it to a screen in a wrong encoding.

The third requirement is also important. The aggregation format should not reduce the bundling benefits by significantly increasing the amount of bytes to transmit.

There are two approaches to solve the problem that are taken into consideration. The first one is based on MIME, also known as MIME encapsulation, and, the aggregation of HTML documents. Another way of solving the problem is to pack several response messages in a raw form into one bundle one by one.

MIME

The MIME standard is used to describe a body of an e-mail message [34]. The predecessor of MIME is the standard of e-mail messages ARPA – RFC822[24]. It was developed to exchange with text messages. Since the publication of RFC822, hardware and software capabilities increased and it became apparent that a lot

of information types that were used in networks cannot be transmitted via e-mail without special tricks. For example, in RFC822 it was impossible to include graphics, audio and video types into a message's body. RFC822 does not provide facilities to transmit text data that is not in the ASCII 7 bits encoding. It is obvious that it is not possible to transfer hyper-text with different styles. Also there are limitations of RFC822 when using different mail systems.

In a sense, MIME is orthogonal to RFC822. The later describes in details the body of a message and the mechanism of its sending. The former is oriented towards the description of the structure of a body and the possibility of its construction from information units of different types.

In the standard several methods of representation of different information types are reserved. For this task special headers of an e-mail message are used:

- the MIME version field that is used to indicate a message represented in this standard;
- the information type description field that provides the right interpretation of data;
- the encoding type that defines the decoding procedure;
- two additional fields reserved for more detailed description of a body.

MIME was developed as an extensible specification assuming that the number of data types would grow with the development of data representation forms. Each new type has to be registered in IANA. This is done in order to control the information types number increasing.

In the following, we describe different MIME headers [35]. The MIME version field is inserted to tell an e-mail client that a message is composed according to the MIME standard. The format of the field is:

```
MIME-Version: 1.0
```

The field appears in the general header of a message and is related to the whole message. It should be noted here that MIME allows to mix headers and bodies of a message. That is why all fields are divided in two groups: general headers that appear in the beginning of a message and private headers that are related only to single parts and appear before a part.

The `"Content-Type:"` field is used to describe the type of information contained in a body. This field tells a client how to interpret the data correctly. The same field is used by a sender to encode the message. MIME defines 7 data types that can be used in the message's body: text, multipart, message, image, audio, video, application. The format of the field is:

```
Content-Type := type "/" subtype *[";" parameter]
type := "application" / "audio"
      / "image"       / "message"
      / "multipart"   / "text"
```

```
      / "video"       / x-token
x-token   := <"X-" + any symbols>
subtype   := token
parameter := attribute "=" value
attribute := token
value     := token / quoted-string
token := 1*<anything except a space, a control symbol or tspecials>
tspecials := [ ()<>@,;:\"/[]?.= ]; they have to be quoted
```

In the following, we discuss some of the types defined by MIME [36].

"Text". This type tells that there is a text in a body. The main subtype of this type is "plain". The notion of a plain text appeared because there is also a marked text that is a text with control sequences and a hypertext that allows to view the text by following hyper-links. Generally speaking, HTML is a special kind of a marked text that is used in the World Wide Web.

"Richtext" is used to describe a marked text that includes tags according to the SGML specification (SGML was the basis for HTML and XML). A tag is a string of symbols that control the representation of a text. There are starting tags ("<...>") and ending tags ("</...>"). As an example of this text the one like this can be given:

```
<bold>Now</bold> is the time for <italic>MIME</italic>
```

A special type of a marked text is described by "html". This is a hypertext. The principles of HTML are the same the same as "richtext". But also there are tags that are introduced to insert hyper-links.

"Multipart". This type defines a mixed document type. It consists of fragments of different types. This type has several subtypes.

- "Mixed" subtype is a message that consists of several beds divided by a boundary that is given as a parameter of the subtype.
- "Alternative" subtype allows to define a variable view of a document depending on a viewer program. E.g. if the viewer can handle only text information or if it can also parse an HTML code.
- "Digest" subtype is dedicated for a multi-purpose message when different parts get more detailed information attached.
- "Parallel" subtype is used to compose a message that has the parts that have to be displayed simultaneously. It is assumed that several viewers are running or a viewer can render multiple resources in parallel.

"Message" type defines an encapsulated message that cannot be transmitted "as is" due to different reasons. The main reason is that it is too large for the underlying e-mail transport facilities. There are 3 subtypes that can be used with this type:

- "Partial" subtype is used to transmit one large message by parts and to re-build it at the receiver.
- Another subtype is "external-body" that allows to refer to some external resources. This subtype resembles the "text" hyper-link.
- The standard subtype of the "message" type is "rfc822". This subtype tells that a message is of the RFC822 type.

The types for the non-textual information. There are 4 major types of that kind:

- "image" to describe graphics. The most common file formats are "*.gif" and "*.jpeg"
- "audio" to describe audio information.
- "video" to transfer movies. The most popular is the MPEG format.
- "application" to transfer the data of any other formats. It is usually used to transfer binary data.

The "Content-Transfer-Encoding:" field. A lot of data is transferred via e-mail in its original form. It can be 7bit symbols, 8bit symbols, 64base symbols and so on. But when a message travels through different e-mail systems it is necessary to set a mechanism of its representation in a standard form US-ASCII. There are encoding procedures for that sort of data. The most widely used is uuencode. In order to decode the data correctly the field "Content-Transfer-Encoding" is introduced. The syntax of the field is:

```
Content-Transfer-Encoding := "base64" / "quoted-printable" /
                             "8bit"   / "7bit"             /
                             "binary" / x-token
```

Each alternative is used in appropriate case. "8bit", "7bit", "binary" do not do any data transformation because e-mails are transmitted by bytes and SMTP does not make any difference between these types. They were introduced in order to strictly separate the data types descriptions. "Base64" is usually used with the "text/ISO-8859-1" type. "X-token" allows a user to specify her own data transformation procedure.

There are two additional fields that are not mandatory: "Content-ID:" and "Content-Description:". The first one is the unique identifier of the content. The second one is used to add comments to the content. They are not displayed by a viewer usually.

We conclude this introduction to MIME by providing a complex example for a MIME usage.

```
MIME-Version: 1.0
From: Anonymous Noname <mail@example.org>
Subject: A multipart example
```

```
Content-Type: multipart/mixed;
              boundary=unique-boundary-1

             This is the preamble area of a multipart message.
             Mail readers that understand multipart format
             should ignore this preamble.
             If you are reading this text, you might want to
             consider changing to a mail reader that understands
             how to properly display multipart messages.

--unique-boundary-1

             ...Some text appears here...
             [Note that the preceding blank line means
             no header fields were given and this is text,
             with charset US ASCII.  It could have been
             done with explicit typing as in the next part.]

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

             This could have been part of the previous part,
             but illustrates explicit versus implicit
             typing of body parts.
--unique-boundary-1
Content-Type: multipart/parallel;
              boundary=unique-boundary-2

  --unique-boundary-2
  Content-Type: audio/basic
  Content-Transfer-Encoding: base64

             ... base64-encoded 8000 Hz single-channel
             u-law-format audio data goes here....

  --unique-boundary-2
  Content-Type: image/gif
  Content-Transfer-Encoding: Base64

             ... base64-encoded image data goes here....

  --unique-boundary-2--
--unique-boundary-1
Content-type: text/richtext
```

```
              This is <bold><italic>richtext.</italic></bold>
              <nl><nl>Isn't it <bigger>cool?</bigger>
--unique-boundary-1
Content-Type: message/rfc822
From: (name in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

              ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

So in general the MIME standard allows to extend the area of e-mail usage and to provide an access to different network resources in standard forms. Also it can be applied in DTNs [8]. It allows to describe resources that are packed in a single packet. This description provides a browser with the necessary information on how to display these resources correctly.

MHTML Review

MHTML is a MIME format based on the MIME technology designated to transmit HTML documents via e-mail [37]. An MHTML message has the MIME structure that consists of a main HTML page and all the embedded resources. It also has a `"Content-Location"` header that provides resources' URIs inside the multipart body and makes it possible for a client to reconstruct the page after unpacking MIME contents.

MHTML is developed to transfer whole web pages in one data pool. It is a good tool to apply it to bundles. Also, it allows preserving HTTP headers. Some of those headers directly map to MIME headers and the ones that do not can be included as extension headers. So, MHTML is fully compliant with the requirements 1 and 2.

When MHTML is used for these purposes (to transmit data via e-mail) the content should be encoded. There is a specification US-ASCII according to which the data is sent by octets that are not exceeding 7 bits. The length of a line cannot be longer than 1000 octets. This restriction means that a binary content has to be encoded. In this case, "binary" means "not 7-bit" and mostly the web content is binary.

HTML pages containing some text usually use "8-bits or more" encoding. Files that mostly consist of US-ASCII symbols (ISO-8859 documents) can be encoded with the `"Quoted printable"` scheme. Files with arbitrary byte sequences (e.g. images) must be encoded with the `"Base64"` scheme. This encoding translates binary data into a text stream that includes only US-ASCII symbols. As a result of such encoding the data volume increases by 33% that restricts the area of MHTML usage.

In our case, this restriction is not applied because the SMTP transport is not used. A binary content may be tagged with the `"Content-Transfer-Encoding"`

header and included directly into the MHTML structure. So the overhead imposed by MHTML is not significant and the third requirement is also met.

Another aspect that has to be taken into consideration is that a HTTP response transmitting resources in MHTML format should have a `"Content-Type"` header that would be useful. In the given implementation the specification RFC822 is used. It is a generic type that is used for MIME messages. More specific content type such as `"message/mhtml"` could be used but from the practical point of view it does not matter though it must be different from `"text/html"`.

Raw HTTP

As an alternative to MHTML a usage of unstructured aggregation is possible. Bundles can contain raw HTTP responses. The responses may be inserted into a bundle one following the other so that the first response is for the first request of the main resource and the following are for the embedded resources. The `"Content-Length"` header may be used to determine resources borders that allows a client to unpack the resources correctly [1].

In order to overcome the problems that appear in the traditional HTTP structure when it comes to DTNs it was suggested to pack the whole page in a single bundle. So a bundle with an HTTP payload contains the whole page, i.e. the page (HTML) and all the embedded resources (e.g. images). And the server does not have to send each resource one by one but it packs them all in one packet and sends to the browser. The browser in turn does not have to send requests for the resources but sends only one initial request, gets the whole page and the resources related to it and then renders the page. The method of this data transfer is shown in Figure 2.

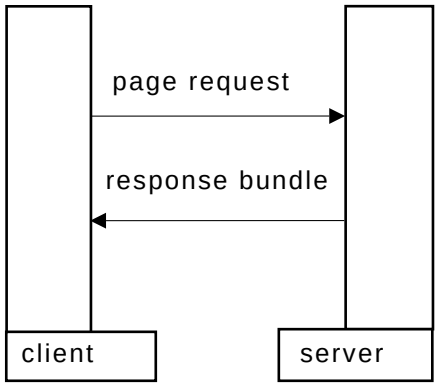

Figure 2: HTTP over DTN scheme

But there does not exist a browser that can work according to this technology. In this work the extension to Mozilla Firefox web-browser is developed that allows to send requests for the whole page, to get and render received bundles. The main idea of the method is that an initial request remains the same but in the response there comes the page with all the associated resources. In this work we have developed the extension for the browser to allow HTTP functionality in DTNs.

4.4 Summary

In this chapter, the main concepts of DTNs are described. The main protocol that is used in DTNs is the Bundle Protocol. Its features allow to use it in challenged environments. There are several ways about how HTTP can be modified to use it in DTNs.

One possible solution is based on MIME. In the response to the initial request a DTN-enabled web server sends not only the requested page itself but also the embedded resources of this web page. The objects are separated using MIME headers. As a MIME-type of such message it is proposed to use `"message/http"` that is used in e-mail systems to send web pages as a whole.

This solution is quite appealing. Firstly, with this approach in an ideal case only one pair of messages is required. Secondly, this is the minimal possible change to HTTP, thus the protocol remains as close to its specification as possible. Finally, compared to using raw HTTP responses and packing them into bundles, the MHTML based approach is standardized and that is why it is used in the extension development.

MHTML approach will be followed for the web browser implementation described in the next chapter.

5 Browser DTN Extension

Based on the modification to HTTP reviewed in Chapter 4 it is possible to create the DTN extension of a browser. It has to solve two tasks: provide the Bundle Protocol as an alternative to TCP and allow to work with aggregated responses of a DTN-enabled web-server.

To avoid creating browser from scratch we utilize the idea of creating an extension to some existing browser. The requirements to this browser include: an open source and powerful extension mechanism. One of the most popular open-source browsers is Mozilla Firefox. It is decided to create an extension to Firefox to enable DTN networks as transport facilities.

Google Chromium is another popular open-source browser. At the moment Firefox has better support for extensions than Chromium. It is possible to create Firefox extensions in C++ whereas Chromium provides only HTML, JavaScript and CSS facilities. It is hardly possible to change anything in the Chromium networking engine without changing its internal structure. At the same time Firefox provides XPCOM technology that allows the deployment of new extensions easily and to change some basic browser's functionality.

In this chapter, the implementation of the extension to a browser is described. The extension allows a browser to work in delay-tolerant networks. The software module is described. Additionally a proxy is developed and implemented. The proxy allows the DTN-enabled browser to fetch web-pages from TCP web servers.

5.1 Browser Structure

Mozilla Firefox is taken as the open-source web-browser to be extended to include DTN as another communication option besides TCP. Firefox uses a COM model as a main system to build a browser. COM stands for the Components Object Model that is used to ease the control of pointers, the garbage collection and the classes instances creation. The COM system used in Firefox is an XPCOM (Cross Platform COM) that creates a run-time environment for the Mozilla execution. That model allows Firefox to run on a machine independently of the operating system that is used [38].

As a main module of networking there is the Necko library. Necko provides different layers of networking: from the presentation to the transport layer. Each protocol type has a module that deals with data of that protocol. For instance, there are nsftp, nshttp, nsfile and some other modules. The Necko architecture is shown in Figure 3.

Necko moves data from one location to another location. These locations are shown in the form of URIs. A URL implements the URI interface. The URI interface performs resource-generic actions like getting/setting the scheme. URLs provide getting/setting of paths, hosts, ports, filenames, etc. URLs are the most commonly used form of a URI.

Generally speaking, Necko provides its own standard implementation of URLs (with class id `NS_STANDARDURL_CID`), that does all of the standard parsing necessary

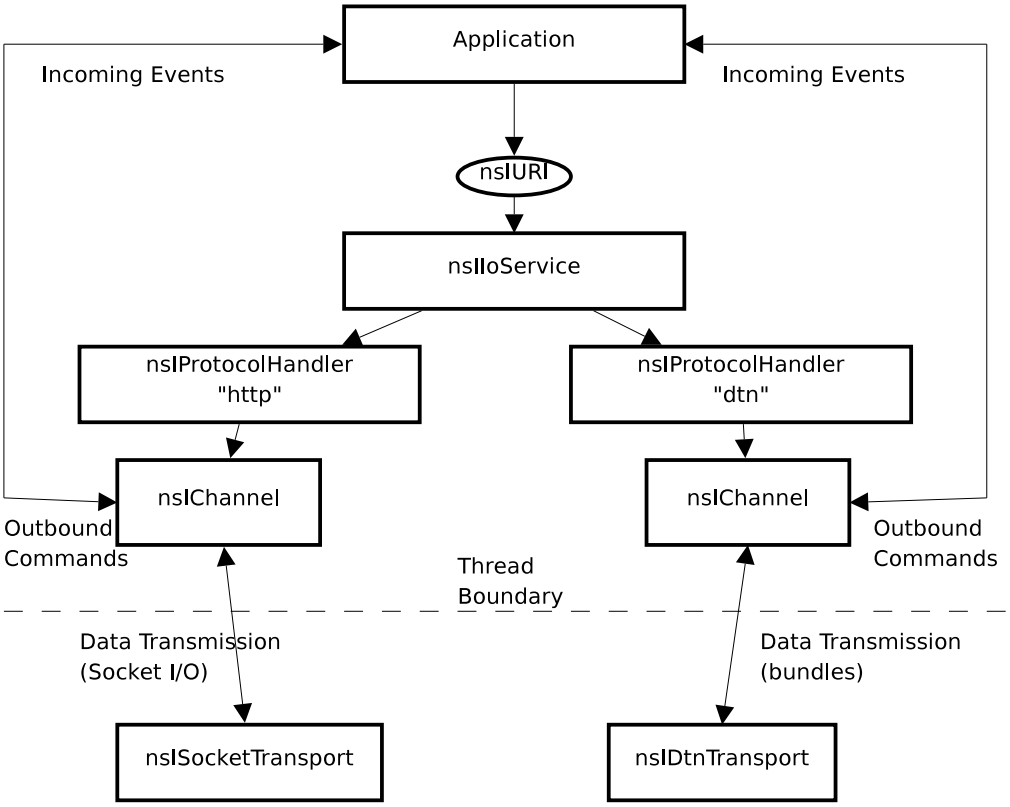

Figure 3: Necko architecture.

for most URLs. Necko also provides a simple URI implementation for convenience. URIs (recall that a URL is just one implementation of a URI) can be created as individual components independently of Necko, or they can be retrieved via the `nsIIOService` interface.

For providing a user's own resource implementation for a new URI type, say `"foo"` URLs, only the `nsIURI` interface needs to be implemented.

In Necko `nsIIOService` acts as the central point of access to URLs and channels (`nsIChannel`). Given a URL string, the `nsIIOService` can be queried for a URL instance representing that string. Once there is an URL interface available (`nsIURL`), there is a completely parsed representation of the original URL string, and it is possible to query for the various parts of the URL.

The `nsIChannel` provides a data access interface that allows reading or writing data to or from a URI. There is a 1-to-1 relationship between URIs and channels. For every URI we want to start the communication with, there is exactly one channel [38].

In order to get a channel from a URI an intermediary layer is introduced. Because a URI has no self knowledge about what "protocol" it represents, another component is responsible for deciding which channel implementation will be responsible for "loading" the URI. This layer is the protocol handler layer. A protocol handler's primary responsibility is to create channels for the scheme it has registered to handle.

For example, the HTTP protocol handler registers itself as the protocol handler for "http". The HTTP protocol handler creates HTTP channels for HTTP URLs.

Once there is a channel data can be read from or written to the URI.

The `nsIIOService` does the protocol handler lookup. In fact, most of what the `nsIIOService` does can be done independently of the `nsIIOService`, in the implementation of the `nsIChannel`. The `nsIIOService` acts as a convenience interface, putting together several steps for the user.

To summarize, the `nsIIOService` creates URL instances. It does this by looking up the protocol handler for the scheme of the URL, then asking the protocol handler to create the URL. The `nsIIOService` can also create channels. It does this by looking up the protocol handler and asking it to create the channel.

The primary responsibility of protocol handlers is to create channel instances that know how to interpret the protocol that the handler registered for. However, a protocol handler implementation also might "cache" the underlying transports so that the channels it creates can reuse them later. A channel performs the main tasks of protocol interpreting and data moving (if any). Although the protocol handler creates channels, protocol interpretation does not begin until the user initiates the transaction using the `nsIChannel` API.

A channel can be read from or written to using either the synchronous API, or the asynchronous API. If data is to be moved asynchronously callbacks are to be received using an implementation of `nsIStreamListener`. An `nsIStreamListener` is told when the URI transaction has "started", when data is available (in the case of reading data), and when it has "stopped". These three events cause the stream to generate notifications that are received by `nsIStreamListener`.

It is up to the `nsIStreamListener` implementation to decide what to do with these various notifications. The `OnDataAvailable()` notification provides a chance for the `nsIStreamListener` to actually retrieve data that the channel has acquired. One of the arguments to the `OnDataAvailable()` notification is an `nsIInputStream` (which can be considered the underlying data).

Necko represents data in the form of streams (`nsIBaseStream` is the root interface. `nsIInputStream` is read from. `nsIOutputStream` is written to). Streams provide a generic interface to the data and allow for many underlying implementations that permit multiple data sources to be used. As a user, we do not care where the data came from, or where it "is", as long as we get the data. The data may be in memory, it may be on disk, or it may be located somewhere else. The point is that streams provide an abstract interface for data access.

If a synchronous transfer on a channel is initiated, a stream that can be read from or written to is handed back. In the case of an asynchronous transfer, callbacks that notify when data is moved, are provided. In the case of an asynchronous `read` when the `OnDataAvailable()` callback is received, an `nsIInputSteam` is handed which we can read the data from (more than likely it is necessary to be able to handle multiple `OnDataAvailable()` callbacks, buffering the received data as necessary). The transaction is not complete until a stopped notification is received.

Underlying I/O functionality has been consolidated into "transports". There is a transport for each type of I/O. Necko implements a file transport and a socket

transport that the protocol implementations (channels) use to actually move bytes to and from.

5.2 Browser Extension

There are two options to start adding the functionality to Firefox:

1. To create a new protocol handler.
2. To modify the HTTP protocol handler.

We choose the first option because adding the bundling services to the HTTP protocol handler is a very complicated task that may be directed to in a further work. A new protocol handler should work with the "dtn:" URI scheme and that is why there is a module created called `nsDtnModule`:

```
#include "nsIGenericFactory.h"}

static const nsModuleComponentInfo gResComponents[] = {
    { "The Dtn Protocol Handler",
      NS_DTNHANDLER_CID,
      NS_NETWORK_PROTOCOL_CONTRACTID_PREFIX "dtn",
      nsDtnHandler::Create
    }
};

NS_IMPL_NSGETMODULE(dtn, gResComponents)
```

As becomes apparent from the above code fragment, a lot of work in the XPCOM world is done by macros, i.e. `NS_IMPL_NSGETMODULE` is a macro that introduces several function implementations that are required for the module to register at the application startup for Firefox to know that now it has a module that can work with the "dtn:" scheme.

The module is a so-called factory. This means that handler class instances are created with the use of the module. The module takes the ownership of them and with the help of COM pointers counts references to them and deletes them when they are not needed anymore.

A handler and a channel are two classes that provide the functionality to read the data from a URI. A handler instance is created and is associated with a URI whenever a new request is created by a user. A channel represents a place where data can be read from or written to.

```
NS_METHOD
nsDtnHandler::Create(nsISupports* aOuter, const nsIID& aIID,
                     void* *aResult) {
```

```
    nsDtnHandler* ph = new nsDtnHandler();
    if (ph == nsnull)
        return NS_ERROR_OUT_OF_MEMORY;
    NS_ADDREF(ph);
    nsresult rv = ph->QueryInterface(aIID, aResult);
    NS_RELEASE(ph);
    return rv;
}
```

The handler creation follows the factory model.

```
NS_IMETHODIMP
nsDtnHandler::NewChannel(nsIURI* url, nsIChannel* *result)
{
    NS_ENSURE_ARG_POINTER(url);
    nsresult rv;

    nsDtnChannel *chan = new nsDtnChannel();
    if (!chan) return NS_ERROR_OUT_OF_MEMORY;
    NS_ADDREF(chan);

    rv = chan->Init(url);
    if (NS_FAILED(rv)) {
        NS_RELEASE(chan);
        return rv;
    }
    *result = chan;
    return NS_OK;}
```

The `NewChannel()` function is used to create a new channel instance. Then the `Init()` of the channel is called. Also a handler has a unique contract ID (CID) in order to distinguish between different protocol handlers (basically between all classes that use XPCOM). This ID should be generated as a Universally Unique IDentifier [39].

```
#define NS_DTNHANDLER_CID     \
  {0x239f0179, 0xa85a, 0x4af0,
  {0xa2, 0xca, 0xc1, 0x46, 0x19, 0x73, 0xac, 0x33}}
```

The `nsDtnChannel` is a class that creates a transport for the transmission. The transport works with a bundle payload and attributes. It owns input and output streams and a pump to work with the streams asynchronously.

The problem solution consists of two parts. It is necessary to create a DTN transport along with a TCP option. And the response that has the `"message/http"` MIME-type has to be processed correctly. The first part is implemented by creating

a socket abstraction to work with the DTN daemon. Then this socket type is used to create the transport instead of TCP sockets. The second part is done through the Firefox converters mechanism. We begin by discussing the first part in more details.

DTN Socket Abstraction

Firefox provides a so-called `"SocketProvider"` mechanism to create sockets of different types. A `"Portable Runtime"` file descriptor is composed of several layers. Each layer is a structure with pointers to the functions like `read()`, `write()` and so on. These functions are called when the descriptor is passed as a parameter to high-level socket functions of `"Portable Runtime"`.

So the creation of a DTN socket implies the implementation of `read()`, `write()`, etc. functions for the DTN layer. But there is a problem that cannot be solved easily. It is described below.

The TCP transport scheme of work is as follows:

1. A domain name is resolved into an IP-address via DNS.
2. Connection is established via `PR_Connect()`.
3. The socket goes to polling loop and when it is ready for reading/writing then `PR_Read()/PR_Write()` is called.

Most of these functions are implemented for a DTN socket in a straightforward way. For example, reading includes bundle reception and sending the bundle's payload piece by piece to the consumer:

```
int nsDtnSocketInfo::Read(char* buf, PRUint32 count)
{
    RecvAll();

    PRUint32 n = mReplyPayload.buf.buf_len;
    if(!n) return 0;

    memcpy(buf, mReplyPayload.buf.buf_val, count);
    if(count < n){
      mReplyPayload.buf.buf_len -= count;
      n = count;
      mReplyPayload.buf.buf_val += n;
    }
    else
      mReplyPayload.buf.buf_len = 0;
    return n;
}
```

Writing includes creation and sending of a bundle. As a connection establishing phase the `dtn_open()` may be used that opens a channel to the DTN daemon. Additionally `dtn_bind()`or `dtn_register()` should be called. `PR_Close()` is mapped to `dtn_close()`.

```
static PRStatus
nsDtnIOLayerClose(PRFileDesc *sd)
{
    nsDtnSocketInfo* info = (nsDtnSocketInfo*) sd->secret;
    dtn_handle_t handle = info->Handle();

    dtn_cancel_poll(handle);
    dtn_close(handle);
    delete info;
    PR_PopIOLayer(sd, nsDtnIOLayerIdentity);
    PR_DestroySocketPollFd(sd);
    return PR_SUCCESS;
}
```

The only problem is DNS resolution. This function does not work with sockets. So the interception of this stage is not possible from the extension. And since there is no DNS service in DTN networks the solution to use `"SocketProvider"` does not work. It is necessary to create the whole DTN transport implementation.

DTN Transport

The task of the transport implementation is quite complicated. Since the DTN API provides only blocking realization for sending and receiving bundles, it is necessary to have a separate thread for the `dtn_recv()` call in order not to block the user interface while waiting for a bundle. There are no nested queues in Mozilla and, therefore, thread and message queue concepts are almost the same, so there used "green threads" where events can be put.

A green thread is a thread that is run by a virtual machine (in case of Mozilla it is XPCOM runtime environment), so it does not use native threads of the operation system. In Mozilla any thread is like a message queue and thus a thread must implement the `nsIEventTarget` interface. Events (e.g. "input stream is ready") are put to this interface. They are scheduled and dispatched by XPCOM.

In general, the code of the DTN transport is the same as Firefox TCP transport that is implemented as a part of the Necko library. The difference is that there is no DNS resolution stage and that instead of TCP sockets DTN sockets are used. The main object of the Necko transport system is `"SocketTransportService"`.

It provides the function to create a transport of a requested type. The transport builds a socket, connects it and then attaches this socket to the transport service that acts as the connections supervisor, too. It is the service that polls the sockets if they are ready to read or write.

Since the service is implemented as a separate thread, networking does not block the Firefox user interface. The file descriptor system with several layers allows the creation of the DTN transport in the same fashion.

To minimize the blocking time of the thread the receiving function call may be involved with a small timeout value. The general `"SocketTransportService"` behavior is to poll the list of sockets. Then `PR_Available()` is called. There is no such function in the DTN API but the implementation is quite simple:

```
static PRInt32
nsDtnIOLayerAvailable(PRFileDesc *sd)
{
    nsDtnSocketInfo* info = (nsDtnSocketInfo*) sd->secret;

    return info->RecvAll();
}

int nsDtnSocketInfo::RecvAll()
{
    dtn_bundle_spec_t reply_spec;
    PRInt32 n = mReplyPayload.buf.buf_len;
    if(!n)
      n = dtn_recv(mHandle, &reply_spec, DTN_PAYLOAD_MEM,
                   &mReplyPayload, 100); //100 - timeout value
    return n;
}
```

To tell if there are bytes available on a DTN "socket" it tries to receive a bundle. If a bundle is received then it is possible to tell how many bytes there are on the socket.

In order to cross the thread boundary and push the data to the main thread an `AsyncInputStream` is used.

```
NS_IMETHODIMP
nsDtnInputStream::AsyncWait(nsIInputStreamCallback *callback,
                            PRUint32 flags,
                            PRUint32 amount,
                            nsIEventTarget *target)
{
    LOG(("nsDtnInputStream::AsyncWait [this=%x]\n", this));

    // This variable will be non-null when we want to call
    // the callback directly from this function, but outside
    // the lock. (different from callback when target is not null)
    nsCOMPtr<nsIInputStreamCallback> directCallback;
    {
        nsAutoLock lock(mTransport->mLock);
```

```
        if (callback && target) {
            //
            // build event proxy
            //
            // failure to create a proxy (most likely out of memory)
            // shouldn't alter the state of the transport.
            //
            nsCOMPtr<nsIInputStreamCallback> temp;
            nsresult rv =
              NS_NewInputStreamReadyEvent(getter_AddRefs(temp),
                                          callback, target);
            if (NS_FAILED(rv)) return rv;
            mCallback = temp;
        }
        else
            mCallback = callback;

        if (NS_FAILED(mCondition))
            directCallback.swap(mCallback);
        else
            mCallbackFlags = flags;
    }
    if (directCallback)
        directCallback->OnInputStreamReady(this);

    return NS_OK;
}
```

An event proxy is essentially the thing that crosses the thread boundary. A `target` here is a thread where events can be put on. A `callback` is an event that will be executed on the `target` thread. `AsyncWait` is executed on the networking thread of Firefox, and the target thread is the Firefox main thread where the data should be pushed. The event proxy allows to start waiting for the data on the networking thread and actually receive the data on the main thread without blocking the main thread in waiting for the network to respond.

As was mentioned earlier, DTN sockets are used instead of TCP sockets. This structure has an advantage because the DTN specific part is separated into a small module. Thus the code maintenance becomes easier. If the DTN API would change it would be quite easy to modify the extension to support these changes.

Another advantage is that the work with DTN is transparent to Firefox modules that are not related directly to networking. DTN is just an ordinary socket and there exists a transport that can be used for DTNs. A request sending remains the same except the usage of DTN transport. The received response has the

type "message/http". Firefox allows processing of this message type by providing converters.

Converter

The task of a converter is to transform an input "message/http" stream into an output stream with a 'magic' type */*. In the original response usually there is an HTML page that may contain references to other resources. This text (as well as in case of TCP transport) is sent to the Firefox HTML parser.

When the parser finds a reference to some external resource, it tries to load this resource via the network. But when the resource is already in the response, because it was packed there by the DTN web server, this process should be interrupted. Instead of trying to fetch the resource via the network the parser should get it from the converter in time when the converter finds out this resource according to its MIME "Content-Location" header.

Because an object of the DtnChannel class is used to load the resource it is possible to stop the loading, if at the moment there exists the converter, because this converter might contain the resource needed. This channel is marked as "waiting for data or ending of the converter's work". If the required resource is not included in the response then it will be figured out after the converter passes through the whole response and finishes its work.

After sending the HTML page to the parser (that may cause the creation of waiting channels) the converter continues its work. It finds a new part in the body of the response message.

It may contain one of the resources that were requested by the parser at the stage of HTML parsing. If this is the case the converter can find the channel that waits for data and that was created for this resource by comparing URIs of all waiting channels with the URI that is defined in the "Content-Location" header of the new part of the "message/http" response.

If the channel was found then it was opened (because channels register themselves as waiting for data in the DTN handler when opening) and it was passed a Listener object that has to be used to send the data to its consumer (in this case the consumer is the parser that opened this channel or to be exact resource loading context that was created by the parser for the resource). This listener is used by the converter to send the found part of the response with the requested URI. Thus the parser gets the necessary resources from the bundle.

If some resources are still missing when the converter is done with processing the message, the channels that are waiting for data at this point go to the network and fetch these resources.

5.3 Proxy Implementation

At the moment, DTN web servers are not available. There is only one testing server running at Aalto. To provide a testing environment for the extensions it is necessary to develop a proxy. The proxy task is to translate between the DTN-enabled web-

browser and the TCP servers of the Internet. Thus the number of web-pages that may be loaded increases dramatically.

Since there is only one server it is impossible to determine the area of DTN-enabled browsers usage. So the task to develop a proxy is important. The main objective of the thesis is to create a DTN browser extension and the proxy is the additional tool that allows to test this extension. To quickly develop the proxy it was decided to use the Haskell programming language since it allows rapid creation of programs and at a reasonable quality level.

Haskell is a pure functional programming language. It means that the main language object is a function. Functions can be pure and impure. Pure functions do not have side-effects. A side-effect is any changing of the world's state, i.e. input/output, global variables reading/writing and so on. The result of a pure functions completely depends on its input arguments and remains the same from call to call. The division of the program code to pure and impure functions simplifies the debugging process, makes the code more readable and, in general, allows the development of software to be faster [40].

There are both Haskell interpreters and compilers. In Linux it is GHCi and GHC accordingly. The fact that a Haskell program can be compiled is an advantage compared to scripting languages. Compiled programs usually perform better. At the same time the proxy was developed mainly to provide a testing framework for the browser. That is why the focus of the implementation was on the correct functioning of the main proxy algorithm and the performance was a secondary problem. It is not assumed that the proxy would be production used.

The proxy operates as follows. In the main program loop it is necessary to wait for bundles from the browser. Each bundle has to be processed like this:

1. Parse the bundle's content to ensure that it contains an HTTP request and to retrieve a URI. The URI is used then in the `"Content-Location:"` field of the response.

2. Send the received request through a TCP/IP network to get the root document of the request.

3. Parse the received document to get the URIs of the external resources that are referred by the root document.

4. Fetch the resources.

5. After getting all the resource pack them into a bundle. The response message format is `"message/http"`, so the resources have to be separated by a boundary and each part's response headers have to be set.

6. Send the bundle back to the browser.

Haskell provides quite a lot of libraries. Using the Network.HTTP library it is possible to implement the TCP part of the proxy. Also it allows to solve subtasks related to parsing of HTTP requests and creating of the final response message. To

perform HTML parsing the Haskell XML Toolbox is used. It includes both XML and HTML parsing modules.

The missing part of the proxy then is a DTN binding. The ForeignFunctionInterface (FFI) is a tool that allows to create bindings from C to Haskell and vice versa. The DTN 2.6.0 implementation is a C-library, so FFI can be used to implement DTN functions in a Haskell program [40].

The work with FFI includes the writing of a module that maps C function calls to Haskell functions. The main task is to convert C types into Haskell types. The DTN implementation introduces some new types to work with DTN, for instance, an endpoint [41]. These types are declared as structures. The work with C structures is quite complicated in FFI (mainly when it is necessary to get access to internal structure fields). That is why a wrapper was developed. The wrapper is a C library that contains some of the DTN related functions and the main work there is to set DTN related structure fields appropriately. The wrapper can be compared to the Firefox DTN socket abstraction that was described earlier but in this case it was necessary to wrap only some functions. For example, here is the wrapped version of the `dtn_register()`:

```
int wrap_register(dtn_handle_t handle, const char* endpoint,
                  u_int exp, dtn_reg_flags_t flags,
                  dtn_reg_id_t* newregid)
{
  dtn_endpoint_id_t eid;
  dtn_reg_info_t reginfo;
  memset(&reginfo, 0, sizeof(reginfo));

  if(parse_eid(handle, &eid, endpoint) == 0)
    return dtn_errno(handle);
  dtn_copy_eid(&reginfo.endpoint, &eid);

  reginfo.regid = DTN_REGID_NONE;
  reginfo.flags = flags;
  reginfo.expiration = exp;

  return dtn_register(handle, &reginfo, newregid);
}
```

As can be seen the fields of the `dtn_endpoint_id_t` structure are set and then the actual call to the `dtn_register()` is performed.

In the following, we discuss the resources bundling strategy in more details. In HTML there are tags that can refer to external resources, for example, <`img`>, <`script`> and so on. For simplicity the proxy bundles images only. It results in additional requests from the browser to get stylesheets, favicons (small icons that are displayed in the navigation bar and in bookmarks near the name of a site) and so on.

In general, the implementation of a proxy that bundles all the necessary resources is quite difficult because the functionality of such a proxy should be the same as the browser's parser module. Additionally, dynamic web pages that are not stored on a server but are built on the fly may introduce further difficulties.

Another difficulties arise when using such technologies as AJAX [20] or Macromedia Flash [22]. They provide the possibility for a user to interact with a server in real time. This interaction results in many "request-response" pairs. Even though it is possible to pass them through the proxy, doing so will break the whole idea of "HTTP over DTN" that the interaction between a client and a server should be reduced to minimum.

Although the problem of determining what resources to put into a bundle is related to the HTTP over DTN topic it is mainly the problem on the server side. Further research is needed.

5.4 Extension Testing

At the time of the extension development there was only one DTN-enabled server. The resources of this server were fetched by the extension without any troubles. To extend the testing framework the proxy was built. Taking into account the limitations of the proxy discussed above it should be noted that the extension work was good at the circumstances. The main idea of reducing the number of "request-response" parse has proved to work. The browser with the extension is able to fetch pages over DTN, to parse the DTN server response and to display the pages correctly.

It was able to load static web pages. Images that were contained there were packed in one bundle and successfully unpacked and displayed by the browser. That means that the concept of web resources bundling and the using of HTTP over DTN works and solves its main task: to reduce the chattiness of HTTP and to allow its usage in delayed environments.

As for the performance measurements it should be noted that the main time when fetching resources through the proxy was the time that the proxy parser spent in finding the resources that have to be included in the bundle. That is because the parser has to build the model of a page and it takes time.

The time that the extension spends in reading and displaying the page is not so big. It could be said that it does not irritate a user and is quite comparable to delays in the case of TCP usage.

More thorough performance analysis follows but it can be said that the user experience is not altered by using the DTN transport instead of TCP.

Quantitative Estimations

In the previous sections the extension development was discussed. It was shown that the DTN-enabled web browser works as expected. In this section quantitative estimates of *how* it works are given. What we are mostly interested in is the comparison of HTTP-over-DTN and HTTP-over-TCP approaches. In particular, they

should be compared when delays, the number of objects on a single page and the page sizes vary.

Firstly, the setup that was used in the measurements is described. Secondly, the collected data is presented and analyzed. Finally, the conclusions about the work of the extension are given.

Setup

To provide the testing environment FreeBSD with `dummynet` were used. Dummynet is a network emulator [42]. Essentially this is a tool that was used to vary delays. Since dummynet was originally designed for FreeBSD and is a standard on this OS it was decided to use FreeBSD though recently there appeared a port of dummynet to Linux [43].

To test the performance of the extension it is compared with the traditional HTTP over TCP approach on different sites. Especially the important characteristics are the number of objects on a single page and the size of a page. To see the difference in DTN and TCP approaches there were created 9 "artificial" pages. The first type pages contain text only. They vary in sizes: 1KB, 100KB and 1MB approximately. The second type pages contain 10 images and they vary in sizes in the same way. Finally, the third type pages have 100 images each.

As a TCP server `nginx` was used [44]. It provides the necessary capabilities and does not require a lot of resources like, for example, Apache. As a DTN web server the modified version of the proxy that was described in the previous chapter was developed.

The data collection process consisted of downloading each page 10 times and varying delays: without `dummynet`, 1 second and 2 seconds introduced on the link by `dummynet`. So that ping statistics in these 3 cases is shown in Table 2. RTT is a Round-Trip Time.

Delay	RTT min	RTT avg	RTT max
0s	0.151	0.164	0.177
1s	1.165	1.174	1.183
2s	2.066	2.153	2.231

Table 2: Clien-server ping statistics, s

It may be assumed that the DTN version of the browser should show better results when delays are long and the TCP version should win in setups with low delays. But basically since the DTN version of the protocol does not introduce any overhead the results for the low delays setup should be comparable.

Results

The measurement results for the case when there was no additional delay introduced on the link are presented in Table 3.

TCP/DTN	1 object	10 objects	100 objects
1 KB	0.14/0.15	0.12/0.15	0.43/0.16
100 KB	0.28/0.22	0.14/0.23	0.43/0.18
1 MB	2.45/0.81	1.35/0.88	0.46/0.80

Table 3: No additional delay TCP vs DTN download times, s

TCP/DTN	1 object	10 objects	100 objects
1 KB	2.02/2.33	3.72/2.17	18.5/2.16
100 KB	4.24/4.10	5.10/5.13	18.4/4.45
1 MB	29.2/30.7	24.6/33.5	20.1/29.5

Table 4: 1s additional delay TCP and DTN mean download times, s

Table 4 contains the results with 1s additional delay on the link.
Finally, the results in case of 2s additional delay are given in Table 5.

TCP/DTN	1 object	10 objects	100 objects
1 KB	6.24/7.19	9.08/7.38	36.9/7.42
100 KB	12.5/15.4	12.9/13.7	36.8/14.7
1 MB	81.0/65.7	65.0/70.3	51.4/62.0

Table 5: 2s additional delay TCP and DTN mean download times, s

Observations

The following observations can be made. First of all, for small and medium sites with 1 and 10 objects TCP and DTN versions show quite the same results. The impact of bundling responses is not visible for these small, in terms of the number of objects, pages.

Secondly, DTN performs better than TCP when the number of objects grows. And this is completely due to the fact that instead of fetching each resource one by one the browser gets the whole bundle related to the requested page at once. The result is visible even in case when there is no additional delay introduced on the link because of TCP way of work and, in particular, because of its feature called "slow start". It may take a lot of time before a new connection starts to use the full throughput of the channel. Thus when there are a lot of connections (instead of one in case of DTN) the time to load the whole page grows.

Thirdly, there are exceptions and somewhat odd numbers in the tables. All of them occurred for big sites. The first strange observation is that TCP performs worse than DTN when there is no additional delay introduced on the link for big sites with 1 or 10 objects. This may be explained by the way of work of Necko

internal buffers. They become a bottleneck and they are actually not used by the DTN extension.

The second observation related to big sites is that TCP shows better results than DTN when the number of objects grows. This is may be due to the fact that the browser fetches the objects in parallel. The `network.http.max-connections` parameter was set to 1 but it seems that Firefox ignored it due to some reason. The number of parallel connections used was 6. Additionally, the pipelining feature should be mentioned. It causes Firefox to send several requests simultaneously without waiting for each response. It also reduces the time to fetch the page with a lot of number of objects.

In general, it should be noted that the DTN download time does not depend on the number of objects on a page. It means that the bundling of resources works as expected. Additionally the DTN extension does not make the situation worse for small and medium sites, so that the performance for them does not degrade.

Internet web pages statistics

Finally, we present the sizes of about 420 web sites on the Internet to determine what is the most common number of objects on a page and what is the average page size. The data is collected from Top500 sites by the web information company Alexa [45]. 80 sites were excluded being different localizations of the same pages.

On average there are 82 objects on a page and table 6 shows the quantile values. The average size of a page is 653 KB and the quantiles are given in table 7. It should be noted that the estimates given above cover about 80% of modern web sites. The fact that the DTN version of the browser has shown good results in this testing tells that the idea of pages bundling can be used to bring the Internet to challenged environments.

Quantile	25%	50%	75%	100%
Objects	23	60	113	507

Table 6: The number of objects on typical web pages

Quantile	25%	50%	75%	100%
Size, KB	231	511	885	4398

Table 7: Internet web pages sizes

5.5 Summary

In this chapter, the extension of a browser was described. The Firefox XPCOM technology provides a powerful framework for extensions development. To add the DTN functionality to Firefox the DTN socket abstraction was implemented. Then

the DTN transport was added as another option beside TCP. Finally, to process the `"message/http"` MIME data type the converters mechanism was used.

To provide a testing framework for the extension a proxy was written. It was developed in Haskell in order to reduce development time. The proxy is mainly designated for testing purposes.

At last the testing of the extension was done showing good results. Instead of several round-trips when using HTTP over TCP only one request-response pair is used in the DTN setup. The development of a DTN-capable web browser is a huge step forward in the problem of bringing HTTP to DTNs.

6 Conclusions

This thesis solves a new and important research problem of the HTTP protocol modification for it to work in delay-tolerant networks. The relevance of the problem is based on the fact that the development of the "space Internet" and networks with the lack of any stable data transmission infrastructure (i.e, floods, global ecological disasters, in deserts and so on) is very important.

We can draw a couple of conclusions. The problem is quite new, the work on the protocol is still ongoing, and there is a serious lack of any literature and specifications.

In this thesis the detailed analysis of the traditional HTTP protocol is done. The cases when there are problems with its functioning are given. The method of its modification is proposed and an extension of an open-source browser with a DTN-enabled protocol stack is developed.

Traditional HTTP cannot be used in DTNs due to the number of round-trips. The scheme that is based on fetching web page resources one-by-one does not work in challenged environments because delays may be long and thus downloading a page according to this scheme results in unacceptable delays.

To solve this problem the resource bundling concept is used. It means that resources are packed in a single bundle. They are sent at once along with the root web page. It allows to fetch a web page in one round-trip in an ideal case. This is crucial to the HTTP functionality in DTNs.

The next steps were done in order to solve the task. The HTTP protocol in TCP/IP networks was analyzed. Its features that are not applicable in DTN environments were shown. Its modification, that is the algorithm changing, was done. The new algorithm was developed and as a part of the Mozilla open-source browser the DTN alternative was implemented. The testing of new methods and modules was done.

The protocol modification solved its main task: to reduce the number of round-trips in order to minimize the page loading time in challenged environments. The usage of MHTML allowed to pack a page with its embedded resources into one bundle and to render them correctly on the client side.

The browser extension was developed to support the new transport facilities in a web browser. It has solved two tasks: to provide the DTN along with the TCP transport option and to process MHTML response in order to render the page correctly. The extension works with `dtn:` URIs. The possibility to work with `http:` URIs was considered, too. Unfortunately, adding this functionality to the extension appeared to be a complicated task and it was not done within the thesis. Adding the possibility to work with `http:` URIs should be one of the future work goals.

The extension proved to work with the DTN-enabled web server and to fetch and display web pages. To provide further testing framework a proxy was implemented. The proxy worked between a DTN-enabled browser and traditional TCP web servers. With the help of the proxy it was shown that the DTN extension is able to fetch static web pages. The problems appeared with loading dynamic web pages and interactive pages based on such technologies as AJAX or Flash.

The future work includes solving the problems with dynamic web pages, adding the functionality to work with `http:` URIs. Also security consideration were not taken into account. The security in DTNs is an emerging area of research by itself, so adopting main DTN security principles to "HTTP over DTN" is left for the further work.

Overall, the solution to "HTTP over DTN" problem and the client side that was developed in the thesis have shown that HTTP can be adopted to DTNs and that it is possible to implement a browser to work in challenged environments.

References

[1] Peltola, L. “Enabling DTN-based Web Access: the Server Side”, Master’s thesis, Helsinki University of Technology, Department of Electrical and Telecommunications Engineering, Espoo, 2008, 72 p.

[2] Comer, D. “Internetworking with TCP/IP, Vol. 1: Principles, Protocols and Architecture”, 5th ed. Upper Saddle River, New Jersey, USA: Addison-Wesley, 2003, 688 p. ISBN 0-13-187671-6.

[3] Ott, J. & Kutscher, D. “Bundling the Web: HTTP over DTN”, WNEPT 2006 Workshop on Networking in Public Transport, QShine Conference, Ontario, 2006.

[4] N4C Project, [Online], [Cited 15 Nov 2010]. Available at: `http://www.n4c.eu`

[5] Wood, L., Ivancic, W., Hodgson, D., Miller, E., Conner, B., Lynch, S., Jackson, C., Curiel, A., Shell, D., Walke, J., and D. Stewart, “Using Internet nodes and routers onboard satellites”, Journal of Satellite Communications and Networking, 2007.

[6] Postellation Project, [Online], [Cited 15 Nov 2010]. Available at: `http://reeves.viagenie.ca`

[7] Drive-thru Internet Project, [Online], [Cited 16 Nov 2010]. Available at: `http://drive-thru-internet.org`

[8] Wood, L., Holliday, P., Floreani, D., and I. Psaras, “Moving data in DTNs with HTTP and MIME: Making use of HTTP for delay- and disruption-tolerant networks with convergence layers”, Workshop on the Emergence of Delay-/Disruption-Tolerant Networks (e-DTN 2009), St. Petersburg, Russia, 2009. Available at: `http://dx.doi.org/11.1109/ICUMT.2009.5345656`

[9] Wood, L. & Holliday, P. “Using HTTP for delivery in Delay/ Disruption-Tolerant Networks”, work in progress as an internet-draft, draft-wood-dtnrg-http-delivery, 2009, 12 p.

[10] Bush, R. & Meyer, D. “Some Internet Architectural Guidelines and Philosophy”, RFC 3439, 2002, 28 p.

[11] Internet World Statistics, [Online], [Cited 09 Sep 2010]. Available at: `http://internetworldstats.com`

[12] OSI, “Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model”, ISO/IEC 7498-1, 1994, 68 p.

[13] DTN Research Group, [Online], [Cited 06 Jun 2010]. Available at: `http://www.dtnrg.org`

[14] Braden, R. “Requirements for Internet Hosts – Communication Layers”, RFC 1122, 1989, 116 p.

[15] Postel, J. “User Datagram Protocol”, RFC 768, 1980, 3 p.

[16] Internet Assigned Numbers Authority, [Online]. Available at: http://www.iana.org

[17] Moya, J. “HTTP Traffic Surpasses P2P Traffic”, [Online], 2007, [Cited 01 Jul 2010]. Available at: http://www.zeropaid.com/news/8855/http_traffic_surpasses_p2p_traffic

[18] Berners-Lee, T., Fielding, R., and H. Frystyk, “Hypertext Transfer Protocol – HTTP/1.0”, RFC 1945, 1996, 60 p.

[19] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616, 1999, 176 p.

[20] AJAX, [Online], [Cited 25 Nov 2010]. Available at: http://www.ibm.com/developerworks/ru/library/wa-ajaxintro1/

[21] Java Applets, [Online], [Cited 25 Nov 2010]. Available at: http://java.sun.com/applets/

[22] Flash platform, [Online], [Cited 25 Nov 2010]. Available at: http://www.adobe.com/flashplatform/

[23] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, RFC 3986, 2005, 61 p.

[24] Crocker, D.H. “Standard for the Format of ARPA Internet Text Messages”, 1982, 47 p.

[25] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication”, RFC 2617, 1999, 34 p.

[26] Dierks, T. & Rescorla, E. “The Transport Layer Security (TLS) Protocol Version 1.2”, RFC 5246, 2008, 104 p.

[27] Scott, K. & Burleigh, S. “Bundle Protocol Specification”, RFC 5050, 2007.

[28] Kevin Fall, “A Delay-Tolerant Network Architecture for Challenged Internets”, SIGCOMM 2003, Karlsruhe, Germany, pp. 27-34, 2003.

[29] Dawkins, S., Montenegro, G., Kojo, M., and N. Vaidya, “End-to-end Performance Implications of Links with Errors”, RFC 3155, 2001, 16 p.

[30] Wood, L., Peoples, C., Parr, G., Scotney, B., and A. Moore, "TCP's protocol radius: the distance where timers prevent communication", Third International Workshop on Satellite and Space Communications, 2007.

[31] Floreani, D. & Wood, L. "Internet to orbit", Cisco Systems Packet Magazine, vol. 17 no. 3, pp. 19-23, 2005. ISSN 1535-2439.

[32] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, 2007.

[33] Peltola, L. "DTN-based Blogging", Special Assignment, TTK Networking Laboratory, 2007. Available at: `http://www.netlab.tkk.fi/~jo/dtn/`

[34] Borenstein, N. & Freed, N. "MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1341, 1992, 77 p.

[35] Freed, N. & Borenstein, N. "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, 1996, 31 p.

[36] Freed, N. & Borenstein, N. "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, 1996, 44 p.

[37] Palme, J., Hopmann, A., and N. Shelness. "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, 1999, 28 p.

[38] XPCOM, [Online], [Cited 29 Jul 2010]. Available at: `https://developer.mozilla.org/en/XPCOM`

[39] Leach, P., Mealling, M., and R. Salz. "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, 2005, 32 p.

[40] Haskell, [Online]. Available at: `http://www.haskell.org`

[41] DTN2 Reference Implementation, [Online]. Available at: `http://dtnrg.org/wiki/Code`

[42] Dummynet, Available at: `http://info.iet.unipi.it/~luigi/dummynet`

[43] Carbone, M. & Rizzo, L. "Dummynet Revisited", [Online], 2009, [Cited 10 Aug 2010]. Available at: `http://info.iet.unipi.it/~luigi/dummynet`

[44] Nginx, [Online]. Available at: `http://sysoev.ru/nginx`

[45] Alexa, [Online]. Available at: `http://alexa.com/topsites`

[46] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges". IEEE Communications Surveys and Tutorials, Vol. 8, Issue 1, pp. 24-37, 2006.