Delay-Tolerant Networking in Restricted Networks

Johannes Morgenroth, Tobias Pögel, Robert Heitz and Lars Wolf

Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig
Braunschweig, Germany
[morgenroth, poegel, heitz, wolf]@ibr.cs.tu-bs.de

ABSTRACT

For security reasons, many companies usually allow only strictly regulated communication, specific system configurations and predetermined software components. Therefore, often only communication over the HTTP protocol is possible, which operates by using a request-response approach. In this paper we present the design and evaluation of a convergence layer for our DTN implementation IBR-DTN using HTTP as underlying protocol. On the remote side we use a web server and a database for the connection management. Through the usage of long-polling we realize a bidirectional communication. In the evaluation, we show the results of bandwidth and latency measurements.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Store and forward networks*

General Terms

Performance, Measurement

Keywords

DTN, Delay Tolerant Communication, HTTP, IBR-DTN, Convergence Layer, DT-HTTP

1. INTRODUCTION

DTNs (RFC 4838) are a promising approach for application scenarios where delays and disruption in the communication can appear. The vehicular communication is a good example where a wide range of heterogeneous networks are available and used where such challenges can occur. For many application scenarios, the need for communication with a backend inside a company is necessary. Backends are usually powerful and versatile servers to make a wide variety of services or data available.

DTNs use a special Bundle Protocol (RFC 5050) for the communication between DTN nodes. This can be set on top of various transport protocols by using special convergence layers. Due to company security policies in accordance with ISO/IEC 27000 et seq. standards, only specific communication paths over insular ports and protocols are permitted. In addition, other security mechanisms such as cookies or single sign-on may be necessary. As a result, the backends are often reached only by using the Hyper Text Transfer Protocol (HTTP). HTTP is based on the request-response principle and is not designed for bidirectional communication. Since a HTTP connection is the only possible communication channel in such a scenario, the bundle protocol must be encapsulated into HTTP to allow communication between DTN nodes. Long-polling allows to realize a bidirectional communication using HTTP.

The paper is organized as follows. In the next section, an overview of related work is given. Section 3 describes our architecture and the following section 4 the implementation. Afterwards, section 5 shows evaluation results. In section 6, we give our conclusions and a brief outlook on future work.

CHANTS'11, September 23, 2011, Las Vegas, Nevada, USA.

2. RELATED WORK

Vehicular communication in challenged networks and the access to the Internet were already considered in several projects like [2]. In these cases, they used multiple gateways along a road to serve Internet-access to vehicles. This approach results in short-time connections and high disruption rates.

The idea to transport DTN data over HTTP has been mentioned previously in [5]. In this draft, the HTTP protocol has been adapted with several non-standard headers to add some specific DTN meta-data, e.g. source and destination endpoint identifier or the content length. Therefore, some features such as extension blocks are not supported. Moreover, no strict differentiation between client and server is made so that each node can initiate and push/request specific bundles to other HTTP-capable DTN nodes. But in many networks, the network address translation (NAT) mechanism is used, thus a connection must be initiated by the client-side only.

Another area of research is the usage of the Bundle Protocol to transport HTTP-requests and -responses. [3] present an approach for a conversion of the different message types and the required components. With the limitation of the HTTP protocol in some parts of the network, additional gateways for the conversion between HTTP and Bundle Protocol are always necessary. In contrast to our approach, the architecture allows a communication without an end-to-end connection by using multiple DTN nodes.

3. ARCHITECTURE

This section presents our architecture for the integration of DTN nodes inside restricted networks where communication is only possible using the HTTP protocol. This is the case if the server is protected by a firewall that blocks all non-HTTP traffic. Thus, it is impossible to connect to it by using a TCP connection with a binary protocol like the bundle protocol. Another challenge are cellular and wireless local area networks, which are often equipped with a NAT mechanism. As a result, clients can not be reached directly from outside; instead they have to initiate each connection. Therefore, in our architecture the client will establish the connection to the server using HTTP.

The architecture is subdivided into two main components as depicted in figure 1. Due to the limitations of HTTP and the request-response principles, a server-client architecture is essential. On the server-side, a module is required to offer the necessary DTN functionality and a HTTP convergence layer to encapsulate DTN bundles into HTTP. Additionally, a database is needed to store meta-data of stored bundles. In the following we name this component *server*, because it do not initiates any connection and just wait for incoming HTTP connections from DTN clients. The client is suited with an additional HTTP convergence layer and performs `GET` and `PUT` calls to the server.

3.1 Protocol

Basically, HTTP is a request-response protocol. In this case, it is used to encapsulate a bidirectional communication between a public server and a client. As with websockets [1], we use a mechanism called *long-polling* to set-up an HTTP connection for bidirectional communication, but place value on less complexity for the required implementation. In fact we use a plain HTTP call, but the response is delayed as long as the server has no data to transmit. Furthermore, the connection stays open until it is interrupted or a predefined timeout has elapsed.

A `GET` request to a server is started with the common HTTP header and a URL containing the peer endpoint identifiers (EID) of the requesting node as parameter. The latter is required for the routing module of the server, which has to choose the bundles to transfer. The server responds with the code "200 OK", sets the content type to multipart/mixed and the transfer-encoding to chunked. Now, the downstream connection is established and ready to transfer bundles to the client. In front of each bundle, a chunked header is placed. It includes the size of the following bundle and control flags for the HTTP connection. The bundle itself is binary encoded as defined in the bundle protocol specification RFC 5050.

Since it is not possible to send multiple HTTP requests in one connection asynchronously, a second connection is required to push bundles from the client to the server. Those bundles are encapsulated into a standard HTTP `PUT` message which is acknowledged by the server in return. According to the previously defined `PUT` request, such a message contains the peer EID encoded as a parameter in the request URL.

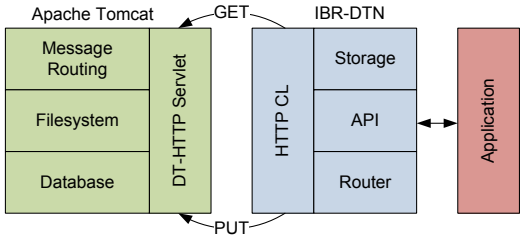

Figure 1: Software architecture

3.2 Client

The client is designed as an additional module in a DTN daemon and shown in the simplified architecture (figure 1). Once the daemon is up, it tries to connect to a HTTP server. On a successful established connection, a `GET` request is sent to the server to poll awaiting bundles for this node. If there are bundles available for the client, it receives them as a continuous stream of data, decode them on-the-fly and pass them to the internal processing of the daemon.

If the client has bundles to send, another HTTP connection is initiated. Each bundle will be encapsulated into an HTTP `PUT` request with the binary encoded bundle in the payload. Thereby, it is possible to transfer all extensions made to the original bundle to the receiving client without being aware of them. Finally, the server acknowledges the received bundle and closes the connection.

3.3 Server

The server consists of the servlet, a database, the routing module and a filesystem to store the binary bundles in files. The tasks are to receive bundles from clients, store and forward them to other clients if they are connected. In addition, it has to consider priorities during the delivery process (Message Routing) and delete expired bundles. If a client provides a bundle to store, the received data is saved into files on the disk. These files are parsed to extract the meta-data of the bundle, which will be stored together with a reference to the file containing the stored bundle in the database.

An incoming HTTP `GET` call contains an identifier of the connecting host encoded into the request URL. This identifier is needed to determine bundles to transfer. Once the connection has been set-up, the thread calls the routing module to request a bundle to transfer. The routing module orders the bundles according to their priority and remaining lifetime before it returns an available bundle. If there is no bundle to be transferred, the thread goes into a wait state and stays there until the network connection is disrupted or another thread notifies it about new available bundles.

4. IMPLEMENTATION

Due to security policies, software components and systems are typically predetermined. For the provision of various services over HTTP, web servers typically exist and can be accessed from outside. In addition, these also have the advantage that they are designed for a potentially large number of connections. Therefore, we decided to use the flexible and scalable Apache Tomcat on the server side in conjunction with a PostgreSQL database which is used for the management of the incoming and outgoing DTN bundles. The encapsulation of the Bundle Protocol within HTTP is realized with a developed servlet for Tomcat and is called DT-HTTP. To extract the meta-data of a received binary encoded bundle, a tool was written using the libraries provided by IBR-DTN. This tool reads the standard input, parses the data

and returns the meta-data of the bundle, which is processed in subsequent.

On the client side, we use our DTN implementation IBR-DTN [4] with an adapted and extended convergence layer which uses the HTTP protocol and does `GET` and `PUT` calls to the server. These are realized with the cURL library, which already supports many features relating to various security mechanisms (like SSL and cookies for single-sign-on) and allows an easy adaptation to different requirements and environments.

5. EVALUATION

To prove the correctness and the performance of the implementation we did tests and measurements with regard to latency and throughput with the HTTP convergence layer. The servlet was deployed on a Tomcat 6, the current stable release of an open-source application server. As platform, a Linux environment was used to run the Tomcat and a PostgreSQL 8.3 database. All machines were connected by a fast Ethernet switch; thus the maximum reachable throughput is limited to 100 Mbit/s. All tests were done in a wired network to avoid inaccuracy and disturbances as expected in wireless networks.

5.1 Latency

In this case, we want to measure the latency of transmitted bundles and set-up two scenarios based on the three machines described above. The first scenario tests the client to server latency. With `dtnping`, a command line tool of IBR-DTN, a bundle is sent to the server, which is reflected by a build-in mechanism and sent back to the client. As second scenario we tested forwarding of bundles from client to another, using the server as intermediate node. Both tests are repeated 1000 times and `dtnping` measures the round-trip time (RTT) of each transmission.

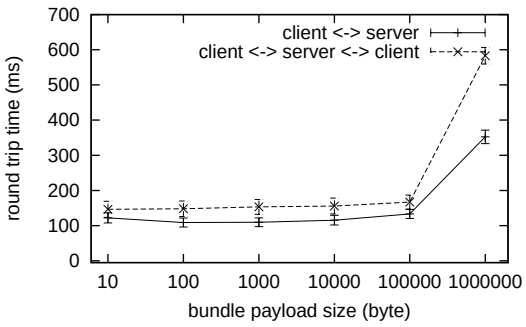

Figure 2: Average RTT and standard deviation

The result of both scenarios are shown in figure 2. For both scenarios the RTT does not seem to be influenced by the payload size if it is lower than or equal to 100 kb. This suggest that the processing delay is more significant in those cases than the transmission delay. In this test, the pass with the very low payload size of 10 bytes got some outliers up to 213.35 ms, which raises the average value to the same level as with 10 kbytes. We assume here a race condition between the thread receiving the bundle and the thread sending the answer. This procedure is stable but rarely leads to a higher delay.

To investigate the correlation of delay and payload size, we took a single transmission of the delay test with an RTT near the average of all transmissions within the same payload size and traced it down with wireshark. We found out, that the significant part of the whole delay is the result of the server processing. In case of forwarding bundles to another client the server needs ≈ 60 ms to process the received bundle. Furthermore, the delay is ≈ 115 ms if the server has to process the bundle to generates an echo, since this is a more complex operation than just forwarding.

5.2 Throughput

Next we measured the maximum throughput. To do this, 1000 bundles with a defined payload size are created by *host1* and forwarded to the server. Then, *host2* is connected and receives the bundles from the server. The test has been repeated with six different payload sizes and we measured the maximum reachable throughput at 90.4 Mbit/s with `iperf` as reference. During the test, the whole network traffic was logged by `wireshark` to get the achieved throughput, bandwidth usage and protocol overhead.

The result of max. 72.9 Mbit/s as downstream performance, the upstream is quite poor with max. 23.5 Mbit/s. This is due to the high processing delay of the server for this manner. To get more details about this delay, we profiled the Tomcat servlet with the *Netbeans Java Profiler*. For each bundle the server receives, a new database connection is established. This delay slows down the whole process. Another reason for the high delay is the processing of the incoming bundle at the servlet. Each received bundle is stored as a file on the disk. Then, the servlet parses the bundle data to extract the meta-data, which is stored in the database afterwards. This store and parse mechanism delays the receiving procedure compared to the sending procedure, which only copies files into the HTTP stream.

6. CONCLUSIONS

In this paper we presented the design and implementation of an HTTP convergence layer by using the long-polling mechanism. This allows a communication between a DTN node and various web services. The present analysis already shows satisfactory results, taking into account the limitations caused by the HTTP protocol.

7. REFERENCES

[1] I. Fette and A. Barth. The WebSocket protocol. Internet-Draft draft-abarth-thewebsocketprotocol-01, Internet Engineering Task Force, Jan. 2011. Work in progress.

[2] J. Ott and D. Kutscher. Why Seamless? Towards Exploiting WLAN-Based Intermittent Connectivity on the Road. In *in Proceedings of the TERENA Networking Conference, TNC 2004*, 2004.

[3] L. Peltola. Enabling DTN-based Web Access: The Server Side. Master's thesis, Helsinky University of Technology, April 2008.

[4] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf. IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation. *Electronic Communications of the EASST*, 37:1–11, Jan 2011.

[5] L. Wood and P. Holliday. Using HTTP for delivery in Delay/Disruption-Tolerant Networks. Internet-Draft draft-wood-dtnrg-http-dtn-delivery-07, Internet Engineering Task Force, May 2011. Work in progress.