A Service Adaptation Middleware for Delay Tolerant Network based on HTTP Simple Queue Service

Hao Zhuang[†‡] , Hervé Ntareme[†], Zhonghong Ou[‡], Björn Pehrson[†]
[†]*Royal Institute of Technology, Sweden;* [‡]*Aalto University, Finland*

Abstract

An increasing number of web-based Delay Tolerant Network (DTN) applications are being developed by researchers, ranging from monitoring the environment to supplementing other network infrastructure located or installed in harsh environments like wireless sensor networks. These applications need to communicate with the DTN daemon to facilitate data transmission. Hence, it is necessary to provide a middleware layer that allows the communication between applications based on different platforms and DTN service daemons. In this paper, a DTN Service Adaptation Middleware (DSAM) is proposed to provide a communication layer between DTN service daemons and different applications based on different platforms (e.g., Java, Python, C/C++, PHP). Furthermore, we delineate the architecture of DSAM and describe two supporting applications based on our middleware, namely DTN2 network management tool and WSN environmental monitoring application. Finally, we present our findings grounded on performance evaluations in terms of throughput and power consumption.

1 Introduction

Even though the mobile telephony and Internet access have experienced rapid improvements over recent years in many developing countries around the world, there are still some rural and remote regions, especially in African countries, which are not equipped with ICT infrastructure. These places are defined as the communication-challenged area [9] where demand for communication services exceeds supply. Moreover, traditional telecommunication operators hesitate to invest in such areas since they only regard them as low-yielding, high-cost and high-risk. In 2003, Demmer et al. [6] proposes the concept of DTN which provides a good solution of communication connectivity for those environment. Later on, DTN2 [7], a reference implementation of the DTN protocols based on Linux C/C++ platform, is developed by the DTN research group whilst Bytewalla [14], developed by Royal Institute of Technology, Sweden, makes it possible for Android phone to become a data carrier for DTN bundles. Android mobile platform is one of the front runners as the DTN mobile router because of its portability and growing popularity.

Meanwhile, an increasing number of DTN applications have been developed to provide a wide range of services for those communication-challenged areas. For example,in [9], DTN network management tool facilitates users to control and manage the DTN2 software while Sentinel Surveillance application improves the health conditions in rural areas. However, these applications are based on different programming languages, such as PHP, Java, C/C++, and are even deployed on different operating systems (e.g. Linux, Android) and hardware. Furthermore, in order to send or receive messages over the DTN, all the applications have to encapsulate their application data units (ADUs) [10] into bundles. It is, in fact, unfeasible to provide different implementations of DTN bundle protocol for different platforms or operating systems, which highlights the necessity of communication between DTN service daemons and various applications.

Therefore, in this paper, we propose a DTN service adaptation middleware (DSAM) to solve this problem. The main contributions of this paper can be summarized in two-fold: i) We propose a DSAM which provides a communication layer between DTN service daemons and different applications. It not only allows application modules to be distributed over heterogeneous platforms (e.g. Java, PHP, and Python), but also reduces the complexity of developing DTN applications that span multiple operating systems and network protocols; ii) Two supporting applications are developed to validate the flexibility, extensibility and reliability of our middleware. On the basis of our performance evaluation, we optimize our system in terms of throughput and power consumptions.

The rest of the paper is organized as follows. In next section,we give background information on both hardware and software. Section 3 illustrates the architecture design of DSAM while section 4 introduces two supporting applications for our middleware. Performance evaluations are presented in section 5. Section 6 gives the related work and section 7 draws our conclusions and puts forward some suggestions for future work.

2 Background

2.1 Hardware

ALIX board computer, designed by PC Engines in Switzerland, is highly power efficient, small, and capable of running operating systems [1] like Linux and FreeBSD unmodified thanks to its x86 compatible processor. Routers, firewalls, mail servers, and network attached storage are all common roles for the Alix. In this paper, Alix boards with Voyage Linux [13] are used as the DTN gateways while Alix boards with Bifrost Linux [5] as DTN routers. The reason why we adopt two different Linux distributions is that Bifrost, a small Linux distribution which follows the principle of KISS (keep it simple and straightforward), provides a robust routing while Voyage Linux offers great convenience for installing and deploying applications on the DTN gateways.

Sun SPOTs (Sun Small Programmable Object Technology) [3] are Java programmable embedded devices designed for flexibility. The basic unit includes application-specific sensors like accelerometer, temperature and light sensors, radio transmitter, eight multicolored LEDs, 2 push-button control switches, 5 digital I/O pins, 6 analog inputs, 4 digital outputs, and a rechargeable battery. Sun SPOTs are powered by a specially designed small-footprint Java virtual machine, called Squawk, which can host multiple applications concurrently, and requires no underlying operating system. It collects light level and temperature data from the environment. Sun SPOT can be tethered to a DTN gateway via a USB cable, which enables them to act as base stations through which other SPOTs can access resources on the gateways such as databases or Web applications.

2.2 Software

DTN2 and Bytewalla are both open source DTN implementation that is compatible with Bundle Protocol Specification (RFC 5050). The goal of DTN2 is not only to embody the components of the DTN architecture, but also to provide a robust and flexible software framework for experimentation, extension, and real-world deployment. In fact, Bytewalla DTN implementation on Android phone is porting from DTN2. It has supported static and PRoPHET routing. The dynamic discovery of the neighbouring nodes enables new nodes to enter and leave the network in a dynamic manner. It eliminates the requirement for a static allocation of host IP addresses and Endpoint Identifiers (EID) to the network nodes as required in the earlier versions of Bytewalla. In this paper, we deploy DTN2 on ALIX board and adopt Bytewalla on Android phones as DTN mobile routers.

HTTPSQS is a lightweight implementation which provides simple queue services based on HTTP protocol [8]. It is very fast and supports ten thousands of concurrent connections. Also, it supports multiple queues, and the maximum length of every single queue is one billion. It takes less than 100MB of physical memory buffer to store dozens of GB data. More importantly, it has only less than 900 lines source code, which makes the deployment and second development become easy. Considering our implementation will be deployed in ALIX Board with limited memory and storage, we choose HTTPSQS as our solution for DSAM.

3 Architecture Design

The middleware we propose is based on the DTN-specific physic, link, network, transport and bundle protocols. Figure 1 shows the architecture of DSAM. As can be seen from the figure, DSAM provides a communication layer that insulates the application developers from the details of DTN bundle protocol, operating system and network interface. The middleware has three distinct parts: (1) HTTPSQS that provides simple queue service based on HTTP GET/POST protocol; (2) Client APIs which enable applications to send and receive different HTTPSQS messages to the HTTPSQS; (3)HTTPSQS Message Processor which processes HTTPSQS messages asynchronously.

HTTPSQS is the core of our middleware which pro-

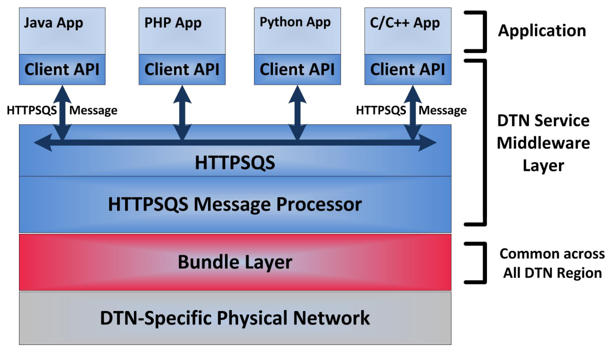

Figure 1: Architecture of DSAM

vides the simple message queue service based on HTTP for the applications. Messages are stored by Tokyo Cabinet [4], as a simple data file containing records, each of which is a pair of a key and a value. Records are organized in hash table, B+ tree, or fixed-length array. It only takes 0.7 seconds to store 1 million records in the regular hash table and 1.6 seconds for the B-Tree engine [2].

Client APIs enable applications to send and receive HTTPSQS messages. It is implemented in different languages according to various applications. At present, we provide PHP, JAVA, Python and C/C++ cross language client APIs for HTTPSQS. It supports two different formats of HTTPSQS Message: text and XML message.

(1)Text message is an ordinary sequential file as textual material without much processing, which is appropriate to run on the ALIX board. The text message is fully compatible with all the commands used in the DTN2 TCL console. For example, the text message to get the DTN bundle statistical information is *bundle stats*.

(2)XML message is stored in plain text format which provides a software- and hardware-independent way of storing data. This makes it much easier to exchange data that can be shared by different applications. However, it is a costly way to analyze the XML string which consumes more CPU time as well as power. In DSAM, XML message is designed to improve the quality of service (QoS). The detailed elements in XML based message are listed in the table 1.

Table 1: XML Elements

Elements	**Description**
messageId	identity number for message
destqueue	the queue which the message will send
expiration	in seconds, which indicates the expiration time, default 60s
priority	range from 1-9, default 4, which indicates the priority of message
timestamp	the timestamp of creating the message
redelivered	the number of message is expected to redelivered
deliveryMode	persistent or non-persistent, which indicates whether the message is stored in the database or not
type	text request reply request-without-relpy. It indicates the type of message body
correlationID	provides provider specific or application specific string help identify filter classify the message
replyTo	The queue which message will reply to
mesg_body	the content of message

HTTPSQS Message Processor (HMP) is main daemon to process HTTPSQS Message asynchronously. All the messages are stored in queues maintained by HTTPSQS. HTTPSQS Message is an object which indicates a specific task. HPM analyzes the messages and dispatches the messages to different applications with different process logic. There are two main components in the HMP.

(1)HTTPSQS Message Serialize/Deserialize HTTP Message is a local representation within a specific application. When it is transferred among different kinds of applications implemented on different platforms, it should be converted from local representation to external representation which is called *serialize*. The reverse process of converting from external presentation to the local is called *deserialize*. In our middleware, we support two kinds of external presentation, namely text and XML, while local representation of HTTPSQS Message uses *class* in an object-oriented design.

(2) Request-Reply Processor Our middleware provides a communication layer for different applications. There are four different kinds of queues created, namely request, reply, retry and dead queue. Under normal conditions, one client application sends a request to request queue and awaits a reply getting from reply queue. If the reply is not sent before expiration of a timeout period, the request will be put in a retry queue and the number of redelivered time is decreased by 1. In other words, under abnormal conditions, the request is not fully finished. If redelivered time decreases to zero, the request will be put in a dead queue for manual process. Meanwhile, a status reply message will automatically generate and be put into the reply queue.

4 Supporting Applications

4.1 DTN2 Network Management Tool

DTN2 starts the DTN daemon service with *dtnd* command on the ALIX boards, which acts as DTN nodes. After startup, *dtnd* creates and initializes a TCL interpreter, listening on the port 5050, which provides a command line interface and an event loop for configuring the DTN2. Through interacting with TCL interpreter, users can monitor the status of DTN2 and also change the setting in a DTN configuration file. For example, *bundle stats* gives the statistical information of bundles while *route dump* lists all of the static routes. For developers, the command line based console is useful for debugging and testing. However, for application users, it is difficult for them to understand all the commands to obtain the status and configuration information of DTN2.

The goal of DTN2 network management tool (DNMT) is to provide a user-friendly management tool for the application end-users, which helps them configure the route

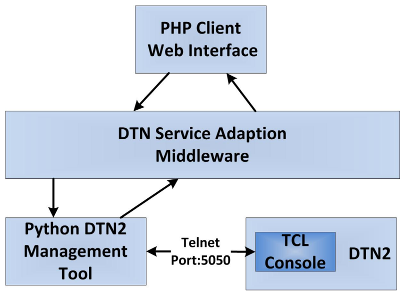

Figure 2: Architecture of DNMT

information and checks the status of bundles and links. Figure 2 shows the architecture of DNMT. In the foreground, users submit their requests through web interface based on PHP and the requests will be sent to the request queue in the form of HTTPSQS message. Meanwhile, in the background, DNMT implemented with Python runs as a service daemon to monitor the request queue. Once there is a message in a request queue, DNMT will get, deserialize and further analyze the messages to understand what kind of task should be executed. The process logic of task is provided by DTN2 service daemon. After that, reply messages will be serialzed and put into reply queue maintained in DSAM. Finally, the reply messages will be refreshed asynchronously to Web interface by using AJAX (Asynchronous JavaScript and XML) technologies. In this application, DSAM provides a communication layer between PHP web interface and Python DNMT to process the messages asynchronously. However, since DTN2 TCL console has already served on the port 5050, we adopt *Telnet* socket communication between DNMT and DTN2 to minimize the response latency.

4.2 WSN Environmental Monitoring Application

Based on DSAM, we also design and implement WSN (Wireless Sensor Network) environmental monitoring application (WEMA), which provides environmental monitoring services for communication-challenged areas. Figure 3 illustrates the architecture design of WEMA. There are four distributed application modules running around the DSAM.

1) PHP web client deployed on DTN gateway is designed to serve as web interface for application users to control the WEMA and DTN services. Through the Web interface, users can send their requests and receive the replies, which make the other applications transparent to the end-users.

2) Java client on SunSPOT deployed on the Sun SPOTs continuously broadcast the environment data (e.g. light level, temperature) on a specific port. The communication between Sun SPOTs and base station is via radio connections following IEEE 802.15.4 standard.

3) Java client on DTN Gateway connecting with a basestation is responsible for receiving the data that is broadcasted by the Sun SPOTs. It analyzes, filters and finally stores the data into the MYSQL database. Moreover, the process logic, which manages the setup and teardown of the connections with Sun SPOTs as well as the allocation of specific sample duration, port numbers and connection timeout, is also included in the client, thereby fulfilling the requests received from the request queue in DSAM.

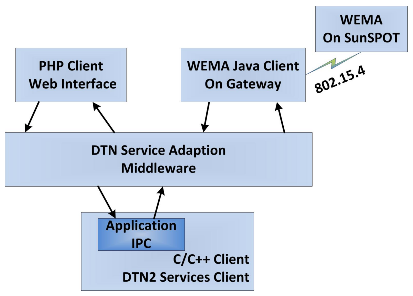
WEMA On SunSPOT
PHP Client Web Interface
WEMA Java Client On Gateway
802.15.4
DTN Service Adaption Middleware
Application IPC
C/C++ Client DTN2 Services Client

Figure 3: Architecture of WEMA

4) C/C++ DTN2 Services Client is to handle the user requests on messages and to filter transmission over DTN regions. There are two pairs of DTN services: *httpdtnsend/httpdtnrecv* and *httpdtncp/httpdtncpd*. The *httpdtnsend* is used to get the messages from DSAM and to encapsulate the messages into bundles while *httpdtnrecv* receives the bundles and puts the messages extracted from bundles into DSAM. With regard to *httpdtncp/httpdtncpd*, *httpdtncp* is used to get file paths from DSAM and to send files as bundles whilst *httpdtncpd* receives the files and returns file paths to DSAM.

We can see that all the application modules only interact with DSAM, which decouples each software component. For each module, they should not know the details of others because they are not depend on each other. Thus, DSAM enables the system continue operation when some part of the system fails, thereby improving the reliability and fault-tolerant of the system. Furthermore, the system has a high level of code reusability and extensibility. For new services such as Healthcare applications, the developers merely need to learn how to use client API to interact with DSAM. Our design frees them from learning the DTN related knowledge.

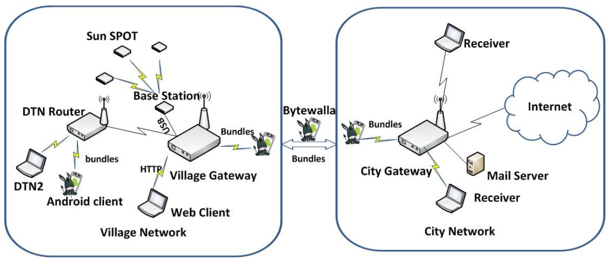

Figure 4: Network Architecture

5 Evaluation

5.1 Network Setup

The figure 4 shows the network architecture of our test network which is used to assess the performance of our DSAM. There are two DTN regions, namely village and city network. Within village network, there is a DTN gateway and two DTN routers. Ten Sun SPOTs are deployed, which are responsible for collecting environmental data. A base station connecting to the DTN gateway with an USB cable receives data from Sun SPOTs. Within every single DTN region, the data are transferred based on the HTTP protocol and are encapsulated into bundles on the DTN gateway, which makes it possible to transmit the data over DTN. Between village and city DTN region, Bytewalla, running on the Android phone as DTN mobile routers, will carry and forward data. Based on this test network, we evaluate the throughput and power consumption respectively from both DTN and DSAM points of view.

5.2 DTN2 Send/Recv Bundles

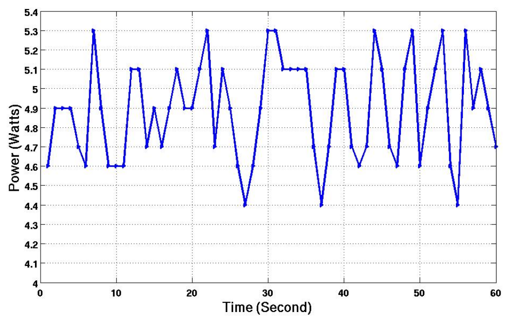

Figure 5: Power of DTN Send/Recv bundles in 60s

We evaluate the throughput between DTN router and DTN gateway. We send/receive bundles of a total 1 Gigabyte data with different file sizes. The size of files are 100MB, 70MB, 50MB, 30MB,20MB,10MB and 1MB. Meanwhile, we record the duration time of the whole transmission. We use *Watt's Up Meter* to get the power during the transmission. Figure 5 depicts that, during a period of 60 seconds,the power ranges from 4.4 watts to 5.3 watts and the average power is 4.89 watts. Due to the limitation of meter, we consider the power remains constant during the process of file transmission. Thus, it follows immediately that the energy consumption is decided by throughput. The higher the throughput is, the less time and energy transmitting the data requires.

The detailed results of different test cases are shown in the Table 2. As can be seen from the table, it only takes 154 seconds for 20MB to finish the file transmission while 1MB needs 338s. From our test cases, the most suitable size of file is 20MB when the throughput is the highest at 6.65 MB/s. That means that we can save half of energy (about 899.8 Joules) if we choose a suitable file size to send data. Therefore, we can divide large files into small ones with the file size of 20MB, which make large savings in energy.

File size(MB)	Duration(s)	Throughput(MB/s)
1	338	3.03
10	194	5.28
20	154	6.65
30	159	6.44
50	177	5.79
70	199	5.14
100	199	5.14

Table 2: Throughput of sending 1GB data in different file size

5.3 DSAM Put/Get Httpsqs Message

For DSAM, we design the test cases to put/get 10,000 HTTPSQS messages with different sizes, namely 32 bytes, 64 bytes,128 bytes, 256 bytes and 512 bytes. Whenever we put or get message we find the power remains constant at 4.6 watts. Figure 6 shows the throughput of DSAM putting and getting messages. It is clear that the throughput of getting messages is larger than that of putting operation. Furthermore, when the message size is 512 bytes, there is a sharp decrease in the throughput of both putting and getting operations. That means the optimal size of HTTPSQS message is 256 bytes to achieve better throughputs, thereby reducing energy consumptions largely.

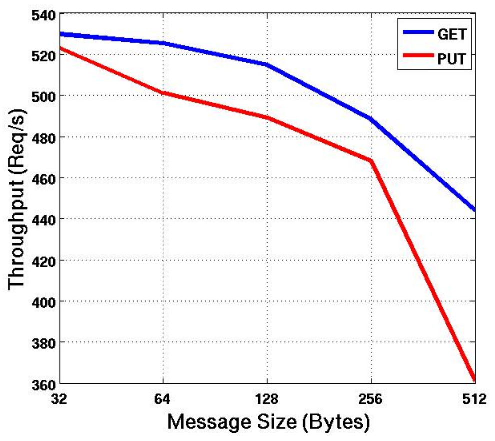

Figure 6: DSAM throughput

6 Related Work

Several studies have conducted on the DTN applications. [9] has integrated Email services and a Sentinel Surveillance health-care application (SSA) for DTN which try to improve the health conditions in village areas of Africa. [11] presents protocol mechanisms for running HTTP-over-DTN as well as a system architecture for incremental deployment. Our design inspiration of DSAM comes from [12] which Agoston et al. propose an adaptive middleware for DTN. However, their middleware provides adaptive connections to seamlessly migrate from one style to another in response to changing network or application conditions while our DSAM focuses on providing adaptation to different applications and services. To our knowledge, there is no other implementation which focuses on the service adaptation between different applications and DTN service daemons.

7 Conclusion

In this paper, we present the design and implementation of DSAM. The unique contribution of DSAM is to create a communication layer between different applications and DTN service daemons, which brings many benefits. Firstly, the application developers are free to choose their development platforms. DSAM provides cross language client APIs for diverse platforms and operating systems, which allows the communication among different applications. Secondly, it is unnecessary for the application developers to understand the details of DTN service daemons. Instead, what they should do is to learn how to communicate with DSAM. Their requests will be processed asynchronously and the reply will be returned to DSAM. Thirdly, the distributed applications based on DSAM can share information and synchronize activity by the HTTPSQS message queuing service, which can achieve better performance, fault tolerance and flexibility. Two supporting applications are also demonstrated the validity and effectiveness of DSAM. Finally, through experiments and evaluation, we optimize the size of files and HTTPSQS messages to improve the throughput as well as power consumption of the system.

Future work can be conducted in following directions. For DSAM, we will add new features such as queue management and event-driven mechanisms to improve the queuing latency. With regard to the applications, we could develop more DTN specific applications based on DSAM. For example, we will design an application that how to connect the DTN region with the social network such as Twitter for DTN based on our middleware. Finally, we would like to conduct more evaluation on the tradeoffs brought by the extra middleware layer.

8 Acknowledgements

The authors would like to thank the Telecommunication Systems Laboratory, KTH for providing research facilities and Bytewalla 5 team for their great contributions. This work was supported by the Academy of Finland, grant number 253860.

References

[1] Alix board. http://pcengines.ch/.

[2] Cabinet benchmark. http://tokyocabinet.sourceforge.net/benchmark.pdf.

[3] Sun spots. http://www.sunspotworld.com/.

[4] Tokyo cabinet. http://sourceforge.net/projects/tokyocabinet/.

[5] BIFROST. http://bifrost.slu.se/.

[6] DEMMER, M., BREWER, E., FALL, K., HO, M., AND PATRA, R. Implementing delay tolerant networking. Tech. rep., 2003.

[7] FALL, K. A delay-tolerant network architecture for challenged internets, intel research. Tech. rep.

[8] HTTPSQS. http://code.google.com/p/httpsqs/.

[9] MARCO ZENNARO, BJORN PEHRSON, H. N. Delay tolerant network on smartphones: Applications for communication challenged areas.

[10] MCMAHON, A., AND FARRELL, S. Delay- and disruption-tolerant networking. *Internet Computing, IEEE 13*, 6 (nov.-dec. 2009), 82 –87.

[11] OTT, J., AND KUTSCHER, D. Bundling the web: Http over dtn. *Proceedings of WNEPT* (2006).

[12] PETZ, A., AND JULIEN, C. An adaptive middleware to support delay tolerant networking. In *Proceedings of the 7th workshop on Reflective and adaptive middleware* (New York, NY, USA, 2008), ARM '08, ACM, pp. 17–22.

[13] VOYAGE. http://linux.voyage.hk/.

[14] YANGGRATOKE, R., AZFAR, A., JOSÉ PEROZA MARVAL, M., AND AHMED, S. Delay tolerant network on android phones: Implementation issues and performance measurements. *Journal of Communications 6*, 6 (2011), 477–484.