BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Information and Communication Systems and Services

A proxy for Military Data Communication over Delay- and Disruption-Tolerant Networks

By: Thomas Halwax

Student Number: 1410258050

Supervisor: Priv. Doz. Dipl.-Ing. Dr. Karl Göschka

Wien, January 16, 2017

FH University of Applied Sciences TECHNIKUM WIEN

Declaration

"As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (for example see §§21, 42f and 57 UrhG (Austrian copyright law) as amended as well as §14 of the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

In particular I declare that I have made use of third-party content correctly, regardless what form it may have, and I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see §14 para. 1 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool."

Wien, January 16, 2017

[illegible]

Signature

Abstract

Joint operations between military forces are the most likely scenario nowadays. To accomplish missions successfully, the exchange of electronic data between heterogenous Command and Control Information System (C2IS) is needed. The Multilateral Interoperability Programme (MIP) defines a Joint Command Control and Consultation Information Exchange Data Model (JC3IEDM) and a Data Exchange Mechanism (DEM) to enable communication without semantic mismatches. The DEM is designed to work on top of UDP and TCP.

When operating in hostile networking environments with long delays and a high number of disruptions, the TCP/IP stack does not perform very well. Delay- and Disruption-Tolerant Networks (DTNs) use message switching instead of packet switching and do not require a predetermined end-to-end path. IBR-DTN is a well supported and well documented implementation for the Bundle Protocol and the Bundle Security Protocol defined by DTN RFCs.

This work provides a transparent, application-aware proxy service for the DEM operating on top of IBR-DTN. All communication phases of the DEM are supported. Considerable effort was needed to make the discovery of DEM nodes work over the DTN. As a result no upfront connection information is needed. The use of DTN group endpoints allows populating mapping tables between the DEM and the DTN address space. These mappings are used subsequently for efficient peer-to-peer communication between DEM nodes over the DTN.

To be platform independent the proxy is implemented in Java employing an asynchronous messaging library.

During the verification process of the proxy all extended features of IBR-DTN are turned off to help focusing on the DEM proxy functionality. All verification steps are executed by a small Javascript test framework running in a virtual environment. The final test was done using production-ready DEMs to ensure compatibility with existing implementations.

Compared to similar approaches, this work does not use general purpose protocols like http or SMTP but focuses on the DEM protocol which is specific to the military domain.

Kurzfassung

Gemeinsam ausgeführte Operationen sind ausgenblicklich das wahrscheinlichste Einsatzszenario für militärische Kräfte. Um hierbei erfolgreich sein zu können, müssen Lage- und Befehlsdaten auf elektronischem Weg übertragen werden. Um semantische Fehler bei der Interpretation der Daten zu vermeiden, wurde vom Multilateral Interoperability Programme (MIP) das Joint Command Control and Consultation Information Exchange Data Model (JC3IEDM) entworfen. Für die Übertragung der Daten sorgt ein Data Exchange Mechanism (DEM), dessen Protokoll sich auf UDP und TCP aus dem IP Stack stützt.

In Netzwerken, die in feindseligen Umgebungen operieren und somit von großen Verzögerungen und häufigen Unterbrechnungen gekennzeichnet sind, gelangt die Leistungsfähigkeit des TCP/IP Stacks an ihre Grenzen. Delay- and Disruption-Tolerant Networks (DTNs) verwenden Nachrichtenvermittlung anstelle von Paketenvermittlung und sind nicht auf einen Ende-zu-Ende Pfad angewiesen, der bereits zu Beginn der Kommunikation existieren muss. Bei IBR-DTN handelt es sich um eine gut unterstützte und gut dokumentierte Implementierung der RFCs des DTN Bundle Protocols und des DTN Bundle Security Protocols.

In dieser Arbeit wird ein Proxy-Service für das DEM Protokoll vorgestellt, das sich für existierende DEM Instanzen transparent verhält und IBR-DTN als Schnittstelle zu DTNs verwendet. Alle Phasen der Kommunikation zwischen DEM Instanzen werden unterstützt, wobei der größte Aufwand in die *Discovery* Phase geflossen ist. In dieser Phase detektieren DEM Instanzen einander über das DTN ohne deren spezifische Verbindungsinformationen kennen zu müssen. Durch die Verwendung von DTN *Group Endpoints* können Mapping Tabellen zwischen den DEM und den DTN Adressräumen erstellt werden. Diese Mappings werden dann für die effiziente Peer-to-Peer Kommunikation zwischen DEM Knoten verwendet.

Um Plattform-unabhängig zu sein, wurde der Proxy unter Zuhilfenahme einer Bibliothek für asynchronen Nachrichtenaustausch in Java implementiert.

Während der Verifikation des Proxy Services wurden alle erweiterten IBR-DTN Funktionen deaktiviert, um den Fokus auf der eigenen Implementierung zu halten. Mit Hilfe eines Javascript Test Frameworks wurden alle Schritte der Kommunikation über den Proxy in einer virtualisierten Umgebung getestet. Zum Abschluß wurde der Proxy gemeinsam mit kommerziellen DEM Instanzen getestet um die Kompatibilität mit existierenden Implementierungen sicherzustellen.

Im Gegensatz zu ähnlichen Ansätzen verwendet diese Arbeit keine Mehrzweck-Protokolle wie http oder SMTP sondern konzentriert sich auf ein Protokoll aus dem militärischen Umfeld.

Contents

1 Introduction

Modern military forces heavily rely on software for making the best decisions for their tasks. To share information about the own and the enemy situation, plans and orders, actions and events et cetera the software needs to communicate with other instances. Since the TCP/IP stack is mature and omnipresent most data exchange protocols for ground forces were designed to run on top of this stack. The *NATO Friendly Force Information* (NFFI) protocol and its successor *Friendly Force Information/Message Text Format* (FFI/MTF) for example can be operated using TCP, UDP or SOAP (on top of TCP). The community driven *Multilateral Interoperability Programme* (MIP) DEM uses UDP for discovery and TCP for the exchange of data.

Besides military head-quaters and non-mobile command posts which are interconnected with high bandwith links (like fiber optics and other hardwired links) ground forces are mobile units (from platoon to section or squad size) that use wireless communication equipment. This may be a modern broadband technology like Long Term Evolution (LTE), Worldwide Interoperability for Microwave Access (WIMAX) or Wireless Local Area Network (WLAN). As a last resort mobile troops may still use narrowband Very High Frequency (VHF) tactical radios because they provide data as well as voice capabilities.

No matter which wireless techology is used, transmitting data on the battlefield is a challenging task. Operating in mountainous terrain or over big distances and facing jamming attacks lead to a high number of disruptions, a high bit-error rate (BER) and thus a substantial packet loss. According to [1] the BER for narrowband VHF tactical radios increases with the distance between sender and receiver. In a rural environment the IP throughput measured in [1] varies between 0,85 kbit/s and 5 kbit/s for a distance of 26 km. For 10 km a maximum of 9 kbit/s can be achieved.

In addition to the low throughput it might not even be possible to establish an IP end-to-end path between sender and receiver. Such paths are required for routing packets. If no path exists routers will drop the affected packets.

Causes for disruptions and thus network partitioning are not restricted to physical or technical reasons. During certain phases of the military intent radio silence (shutdown of the sending part of the radio) can be commanded.

Mobile Ad-Hoc Networks (MANETs) are one possible solution for highly dynamic topologies. They usually operate on IP based networks like LTE, WIMAX or WLAN. Nodes participating in a MANET do not rely on existing infrastructure (like base stations or access points). Each node can communicate directly with other nodes and acts as a router. It forwards packets to destinations which are not within the transmission range of the node. Neither network nor transport layers designed for infrastructure based networking behave well in MANETs and need enhancement. [2] provides a very good insight into MANETs and especially the routing protocols used.

It is the nature of TCP to ensure reliable end-to-end communication. Each packet must be acknowledged and retransmitted in case of an error. While this is not a problem in well connected environments it indeed is for wireless environments with a high error rate and a low or asymmetric bandwidth. Packets may be delayed and the virtual TCP connections may get disconnected in consequence of disruptions.

In order to handle long delay and a high number of disruptions the concept of Delay- and Disruption-Tolerant Networks was introduced as an *internetwork of challenged internets* by [3] based on the idea of an *Interplanetary Internet* [4]. DTNs do not use virtual connections (as TCP does) but are based on an asynchronous messaging pattern. They operate as an overlay network on top of existing protocol stacks. Application data is aggregated into messages which are called bundles. These bundles are forwarded by bundle routers within the DTN using a store-and-forward strategy. A number of routing algorithms are available ranging from flooding to more sophisticated protocols like PRoPHET [5].

1.1 Problem definition

The MIP is a consortium that promotes interoperability between C2IS. Mulinational joint operations are common nowadays and the MIP defines a standardized Joint Command Control and Consultation Information Exchange Data Model (JC3IEDM) and a Data Exchange Mechanism (DEM) for exchanging data between national C2ISs. The DEM relies on TCP and UDP and thus does not work properly in challenging wireless networks.

DTNs are one possible way to cope with these challenges. This work makes use of an existing DTN implementation and aims to create an application-aware proxy service for the DEM. The proxy will be completely transparent so no changes need to be made to existing DEM implementations. Since the DEM is a peer-to-peer protocol the proxy must also support the manadatory node discovery process. Compared to other protocols used for tactical data communication this is a very uncommon requirement since all of them require upfront connection informations (like IP address and port). The design and architecture for the proxy is presented in chapter 4, the implementation details are described in section 4.3. To verify the implementation assertions for a Javascript test-framework are introduced in section 4.4. Production-ready DEM instances are used to finalize the verification cycle.

2 Mulilateral Interoperability Programme

Working together in joint operations - wether civilian or military - is a key factor to successfully accomplish the assigned tasks. This can only be achieved by an unobstructed electronic communication between the involved parties. The interoperability between heterogenous informations systems is a major requirement. To create a common understanding about specific data and to deliver this data to the participating parties in time is extremely important.

The MIP aims at solving data exchange mismatches between heterogenous national C2ISs. The MIP is an organization whose 29 members are nations as well as the North Atlantic Treaty Organization (NATO). Since its founding in 1998 a mutual and mandatory data model (JC3IEDM) and a Data Exchange Mechanism (DEM) have been introduced and refined continuously. The MIP focuses on the development of specifications and does not implement software itself.

Currently, version 3.1.2 of JC3IEDM and DEM is released and most of the members of MIP have successfully fielded their implementations. To ensure interoperability and increase the quality of their implementations, the Coalition Warrior Interoperability eXploration, eXperimentation, eXamination, eXercise (CWIX) is held on a yearly schedule. Running scenario tests similar to real-world experiences (like heavy load, long running tasks, etc.) has led to a stable and mature software and creates new requirements for the upcoming releases of the JC3IEDM and the DEM.

Version 4.0 of the specifications is currently under development and expected to be released in 2017.

2.1 Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM)

The JC3IEDM is a very strict data model designed for the exchange of information between C2ISs. By utilizing the model and its semantics, misinterpretation of data can be widely avoided. It supports a broad spectrum of information areas necessary for civilian and military operations, including data about the own and the enemy situation, operational data, environmental conditions and (geo)political situations.

The model is designed to support only *add* and *read* operations, neither *update* nor *delete* operations are allowed. During missions each addition is recorded and can be used as a warfighting diary for subsequent analysis.

2.2 Data Exchange Mechanism (DEM)

Data within the model can be grouped by using contexts and Organisational Information Groups (OIGs). These OIGs are the foundation for transmitting data belonging to a *Responsible Organisation* between parties involved using the DEM. A DEM schema (DEMS) instance is the origin of OIGs and is hosted by a *replication node*. Figure 1 shows the key entities and their relationship.

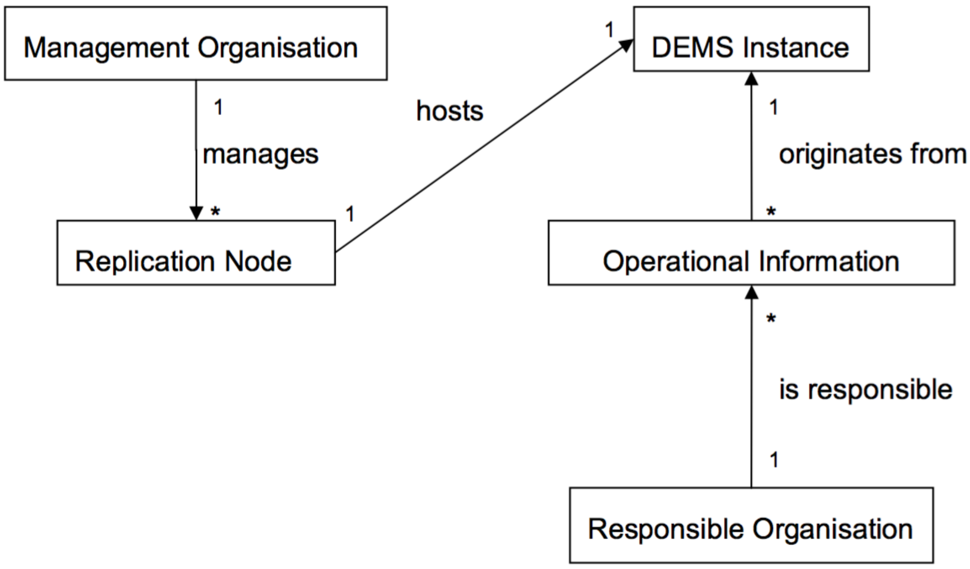

Figure 1: Entities involved in replication data using the *Data Exchange Mechanism* (Source: MIP Technical Interface Design Plan p. B-10 [6], by courtesy of the MIP Programme Management Group).

The primary pattern for the exchange of data is *publish/subscribe*. A set of *available OIGs* is provided by replication nodes upon connection which may be subscribed by other nodes. Once one or more OIGs are subscribed, the providing node (Data Provider (DP)) sends batch updates to the receiving node (Data Receiver (DR)) to bring the DR up-to-date. Whenever there is a subsequent change to the OIG the DP will replicate these changes to the DR. When the DR detects faulty data it will notify the DP and cancel the subscription. Since the *responsible organisation* is the only entity that is allowed to add data to the OIG, the flow of data is always from the DP to the DR.

The DEM is designed to operate on top of the TCP/IP stack and uses two additional layers for handling Protocol Data Units (PDUs). The Transport Manager (TMAN) is directly connected to the underlying TCP interface of the operating system and manages connections between replication nodes. For the identification of replication nodes, a unique *node id* (a nine digit number) must be assigned to each node. The TMAN is also responsible for compressing the data by using the *zlib/deflate* algorithm. The purpose of the Data Manager (DMAN) is to manage OIG subscriptions and to create update messages for these subscriptions. It also provides an interface for the application layer.

2.3 DEM Connection Information (DCI)

Before a TMAN connection can be established, some basic information about other replication nodes must be provided. The DEM Connection Information (DCI) has a simple XML data structure with the following properties:

Scope May either be ANNOUNCE or REPLY.

NodeId A unique nine digit identifier. The first three digits are a prefix that refers to nations or organisations. E.g. the prefix range 120 . . . 129 has been assigned to Austria. The following six digits can be employed by the nations/organisations to ensure uniqueness.

ReplicationNodeIPaddress The IPv4 address of the replication node. The current version of MIP/DEM does not support IPv6. (DNS) hostnames are not allowed.

ReplicationNodePort The TCP port on which the TMAN of the replication node is listening.

ResponsibleOrgName Identifies the origanisation that is responsible for the replication node.

RoleName An optional field to hold information about the role within the organisation.

MipReleaseVersion Major and minor version number of the supported MIP/DEM specification.

Optional Extension An optional container for additional information.

There are three ways of supplying this data:

- a user interface
- an XML file from the file system
- via UDP broadcasts to port 13152

The latter is a very common way to discover replication nodes participating in the local network. Utilizing UDP broadcast, DEM instances send a *DCI ANNOUNCE* on the local network to port 13152. All DEM instances that have received announcements shall send a *DCI REPLY* to the sender using UDP unicast. This sequence shall be processed each time a replication node is started or upon user request.

It remains a national issue to additionally implement discovery using UDP unicast. Since broadcast is restricted to local networks (broadcast domains) DCI announcements are sent to a given IP address via unicast. The receiving node replies using UDP unicast, too.

When one node needs to update its DCI it must close all existing connections to other nodes first. Then the updated DCI must be re-broadcasted or re-sent to the known hosts.

Upon reception of a DCI the replication node must obey the following *business rules*:

- If the NodeId of the sender is not known the DCI shall be considered as *new*.

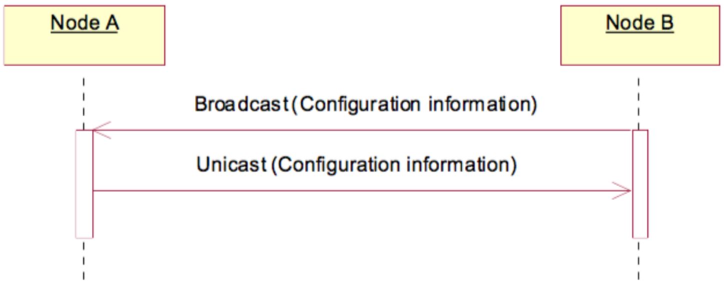
Node A
Node B
Broadcast(Configuration information)
Unicast (Configuration information)

Figure 2: Sequence of operations used for DEM discovery. Node A broadcasts its DCI and Node B replies using UDP unicast. (Source: MIP Technical Interface Design Plan p. B-90 [7], by courtesy of the MIP Programme Management Group)

- If the NodeId of the sender is already known by the receiver but there is no active connection between the nodes the existing DCI is to be *overwritten*.
- If the NodeId of the sender is already known by the receiver and an active connection is currently established between the two nodes the incoming DCI must be *ignored*.

2.4 Transport Manager (TMAN)

2.4.1 Architecture

The *Transport Manager* (TMAN) layer operates on top of the TCP stack. Its purpose is to handle TCP operations provided by the operating system and to manage the lifecycle of a connection between two replication nodes (open, transfer data, close). Figure 3 shows the layered architecture of the DEM.

2.4.2 Protocol Data Unit (T_PDU)

The formal definition of the TMAN protocol PDUs is accomplished using the extended Backus–Naur form (EBNF). See table 1 for details. The payload of a *T2* message is omitted because it is out of scope of this work. For details about the DMAN PDUs refer to the MIP Technical Interface Design Plan section 3.8.2 p. B-53 [6].

Each message is prepended with the length of the message itself. The length of the most simple *T_CHKCON* message for example is `00000002` (bytes).

T_OPNRQ This is the initial message after establishing the TCP connection. It contains the NodeId of the originating replication node as well as the NodeId of the destination node and will only be sent once per connection. For example:

```
00000022T1|123000111|123000999
```

T_DATA These messages encapsulate the DMANs PDUs and are used many times during the lifetime of the connection.

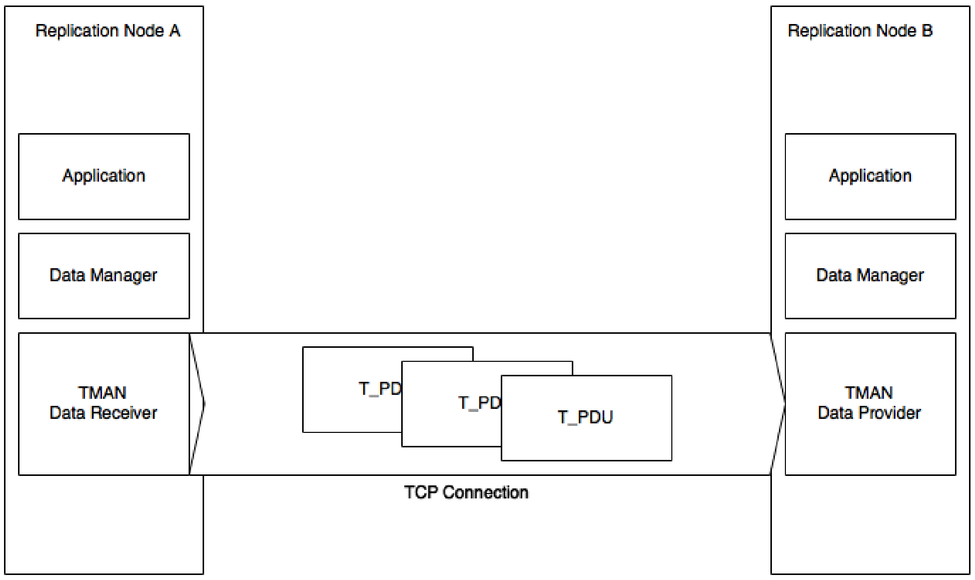

Figure 3: Architecture of the DEM layers showing Node A acting as a Data Receiver and Node B having the role of the Data Provider.

<T_PDU>	<Msg_Length> <T_MSG>;
<T_MSG>	<T_OPNRQ> ∣ <T_DATA> ∣ <T_ABRT> ∣ <T_CHKCON>;
<T_OPNRQ>	”T1∣” <Orig_Node_id> ”∣” <Dest_Node_id>;
<T_DATA>	”T2∣” <DATAMGR_PDU>;
<T_ABRT>	“T3∣“ <Abort_Reason_Code>;
<T_CHKCON>	“T4”;
<DATAMGR_PDU>	** Data Manager PDU omitted **
<Msg_Length>	<padded no. (8)>;
<Abort_Reason_Code>	<padded no. (3)>;
<Orig_Node_Id>	<padded no. (9)>;
<Dest_Node_Id>	<padded no. (9)>;
<digit>	“0”∣“1”∣“2”∣“3”∣ “4” ∣ “5” ∣ “6” ∣ “7” ∣ “8” ∣ “9”;
<padded no. (3)>	3 * <digit>;
<padded no. (8)>	8 * <digit>;
<padded no. (9)>	9 * <digit>;

Table 1: The grammar for TMAN PDUs in the EBNFs.

T_ABRT Abort Messages inidicate a TMAN protocol error where the TMAN cannot recover. Since there is also a business rule to allow only one connection between two replication nodes this message is also used to deny duplicate open requests.

```
00000006T3|500
```

T_CHKCON TMAN messages are asynchronous and there is no agreement about timing. To allow an early detection of broken TCP connections, check messages are sent on a regular schedule. The recommended minimum interval is one minute.

```
00000002T4
```

2.4.3 Connection lifecycle

Node A holds the role of a Data Receiver, while Node B is the Data Provider. The TMAN on Node B listens on a TCP port for incoming connections. The TMAN on Node A initiates a TCP connection to Node B. After establishing the connection Node A starts the TMAN conversation by sending a *T_OPENRQ* message to Node B. Node B will accept the connection and the request considering two *business rules*:

1. The node is willing to exchange data with Node A and does have a local configuration for Node A.
2. There is neither an existing connection nor an ongoing connection setup process to the same Node.

If any of these rules is violated Node B will terminate the connection sending an appropriate *T_ABRT* message (see table 2 for error codes and their semantics).

Figure 4 shows the sequence of operations used during an accurate communication procedure.

2.4.4 Error codes for *T_ABRT* messages

When a TMAN instance detects a protocol error, violations of TMAN business rules or the DMANs indicates an error, the TMAN will send a *T_ABRT* message and shut down the TCP connection. The reason codes and their semantics are show in table 2.

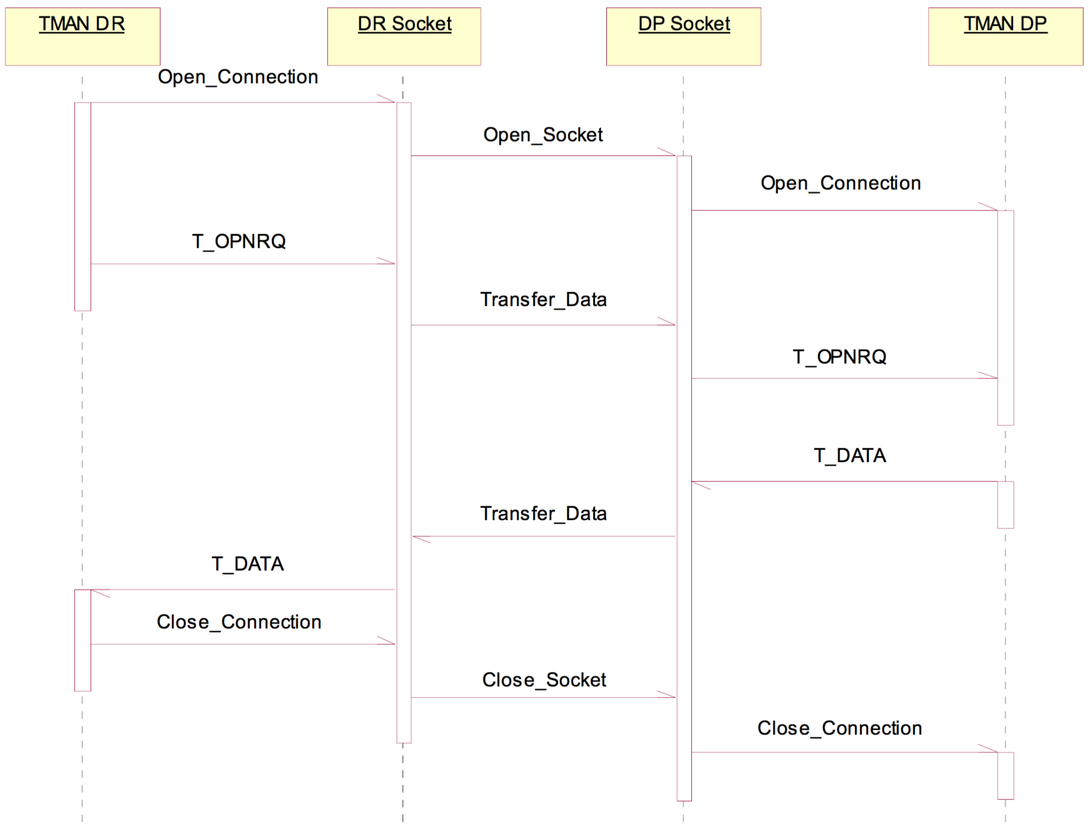

Figure 4: The sequence of operations invoked to open a connection between two replication nodes, transfer data and close the connection on a regular basis. (Source: MIP Technical Interface Design Plan p. B-71 [6], by courtesy of the MIP Programme Management Group)

Code	Reason	Description
500	identification error	The *T_OPENRQ* contains an unknown source or destination NodeId. If the sender's IP address does not match the one provided in the DCI, the receiving node will also abort the connection and use this code.
501	parsing error	The syntax of the *T_PDU* does not follow the grammar (see table 1).
504	duplicate socket	Only one incoming connection between two Nodes is allowed.
506	compatibility error	The DPs does not support the MIP version used by the DRs.
507	unexpected message error	The TMANs as well as the DMANs maintain a finite state machine for their internal state. This error code is used to indicate an error following the correct protocol sequence.

Table 2: Error codes and their semantics for TMAN *T_ABRT* messages.

3 Delay- and Disruption-Tolerant Networks

3.1 Concepts

Despite all efforts to provide infrastructure-based communications for mobile clients there will always be circumstances where network clients face long delays and disruptions. Some applications may seem exotic like transmitting data to a spacecraft orbitting planet Mars. Others, like military operations in hostile environments or search and rescue forces after natural desasters will more likely have to deal with limited networking capabilities.

While IP and TCP based applications perform very well in fast and well connected infrastructure-based networks, they do not in challenging network environments. In [7] more than thirty single proposals are discussed to enhance the performance of TCP in MANETs, but no "one does it all" solution has been identified.

To evade the problems the IP stack has when operating in challenged environments the idea of an *Delay Tolerant Message Based Overlay Architecture* has been introduced in [3]. Inspired by the way e-mail works, data is transmitted using messages ("bundles") instead of (TCP) packets. In contrast to IP routing, no prior end-to-end path must exist. Bundles are forwarded by peers using a store-carry-forward strategy. Thus, delays by the magnitude of hours or even days are made possible. To make sure bundles get delivered to their destination routing protocols for peer-to-peer forwarding (like PRoPHETv2 or GARP2) have been designed.

Many applications trust in the ability of TCP to deliver data in a reliable way. Similar to the acknowledgement in TCP, nodes participating a *Delay- and Disruption-Tolerant Network* (DTN) can send messages to the originator when passing bundles to another routing node. This concept is called *custody transfer*. For signaling transfer to the final destination, a *delivery confirmation* message can be requested by the sender.

Addressing within the DTN employs URIs for *sources* and *destinations*. This flexible concept can easily be adopted for many kinds of applications.

RFC 5050 [8] describes the protocol, the bundle format and all abstract service descriptions that are used in DTNs. RFC 6257 [9] extends the bundle protocol specification and introduces security capabilities. Providing *integrity* and *confidentiality* at least two out of three aspects of information security are addressed. Both RFCs are not part of the *Internet Standard Track Specification* and still in experimental state.

Figure 5 shows the layered architecture in comparison to the IP stack. The interface between the bundle layer and the transport layer is handled by a *convergence layer*. The transport layer may be TCP or UDP but may also be e-mail, Bluetooth, ... and is not restricted to the typical layered hierarchy. As shown in figure 6 the *Bundle Protocol* is the only abstraction required

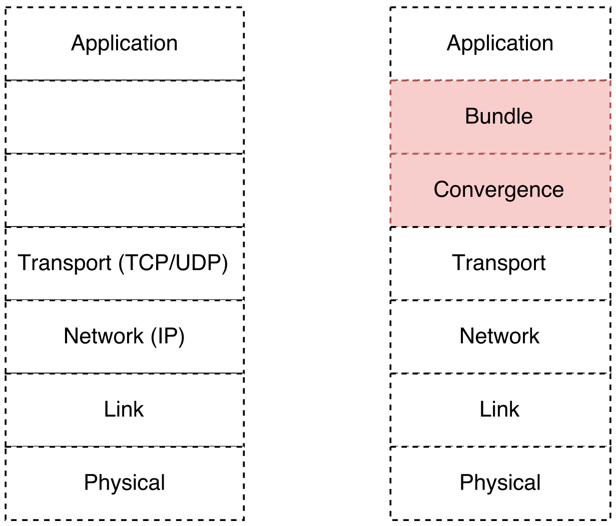

Figure 5: Comparison of protocol layers between the IP stack and the DTN protocol stack (Source: Delay- and Disruption-Tolerant Networks (DTNs): A Tutorial [10])

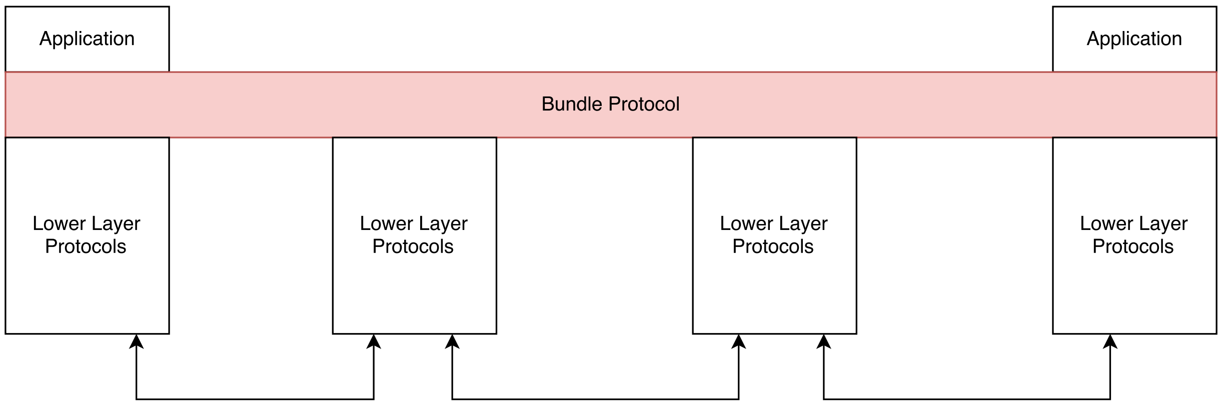

Figure 6: The *Bundle Protocol* is the only layer required for nodes participating a DTN (Source: Delay- and Disruption-Tolerant Networks (DTNs): A Tutorial [10])

across different nodes. This enables even embedded nodes without a full-blown TCP stack to participate a DTN.

This tutorial [10] provides a very good introduction into the basic concepts of Delay- and Disruption-Tolerant Networks.

3.2 Implementations

A lot of work has been done in recent years to bring life into the concepts defined in the RFCs. The reference implementation *DTN2* presented in [11] focuses on experimental routing algorithms. According to [12], it barely uses exception handling and is restricted to specific platforms since it makes use of their features.

The *Interplanetary Overlay Network* (ION) is optimized for space related scenarios and not suitable for networks with opportunistic contacts.

Morgenroth's *Event-driven Software-Architecture for Delay- and Disruption-Tolerant Networking* (IBR-DTN) [12] is a platform independed, well structured and documented software. Written in C++ it provides an plug-in mechanism for routing protocols and convergence layers. It was the first implementation that supports the complete *Bundle Security Protocol*.

Available at github under a permissive MIT license it is still maintained and has an active user group.

4 A proxy for MIP/DEM over IBR-DTN

4.1 Objectives, assumptions and non-goals

The primary objective of this work is to provide an architecture and an implementation for an application aware proxy server for the MIP DEM protocol described in section 2.2. The artifact created can be seen as a proof of concept of operating the DEM over a Delay- and Disruption-Tolerant Network. A well documented implementation of the *Bundle Protocol* was provided by [12] (IBR-DTN) which will be the basement of this work. In order to meet the most important prerequisite, the proxy service needs to be completely transparent to existing DEM implementations. No changes to behavior during discovery or normal operation will be required. This also implies that the proxy does neither have knowledge about the inner state of the proxied TMAN instance nor any business rules to follow.

To make sure to focus on the primary objective all value added features provided by *IBR-DTN* are disabled. Only two instances of the proxy service are involved each using exactly one corresponding *DTN daemon*. The convergence layer uses reliable TCP connections and forwards bundles based on static routing tables. Using a virtual environment also eliminates networking problems.

These assumptions on the DTN environment might not meet realistic szenarios but exclude all interfering parameters.

No attention will be given to performance considerations. The artifact created works *as-is* and does neither optimize speed, memory usage nor any other resource related footprint.

4.2 Architecture

The high-level architecture of the environment used in this paper is shown in figure 7. The proxy is designed to support multiple DEM instances but only two are used in the subsequent explanations and sequence diagrams. Table 3 shows the names, NodeIds and roles of the DEM instances.

Details about the DTN daemon implementation are omitted. The way the DEM DTN proxy interacts with the daemon is not important for the high level architecture and remains an implementation detail (see 4.3).

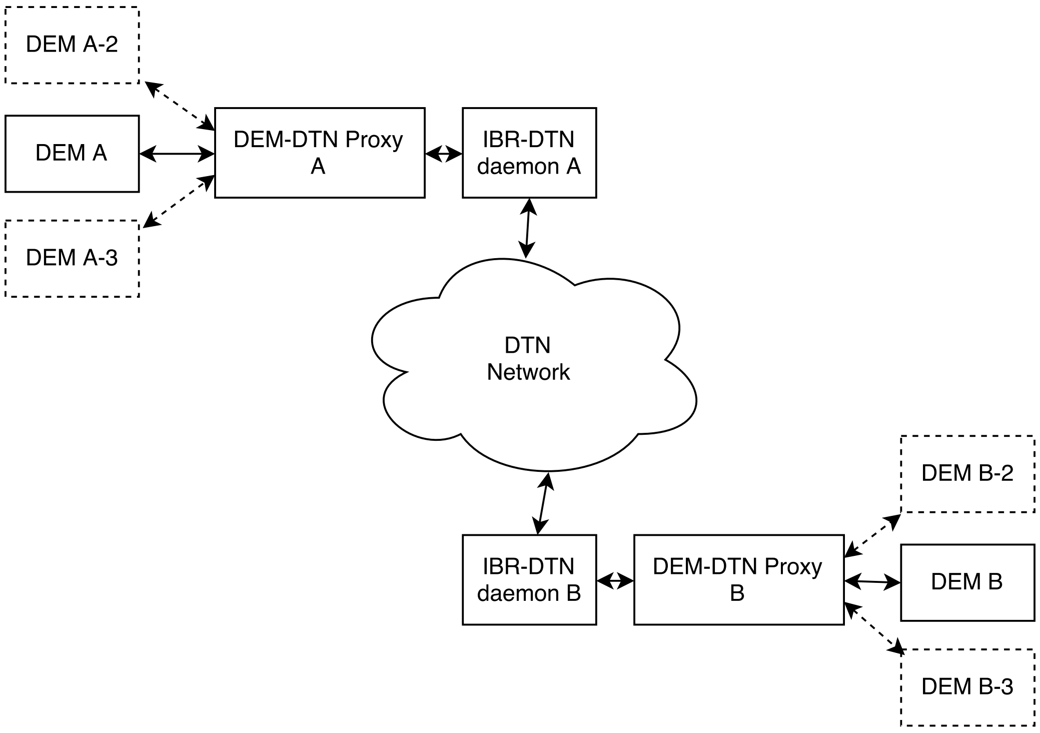

Figure 7: The high-level architectural environment used for the DEM DTN proxy. Objects that are supported but not used in this set-up are drawn using dashed lines.

Name	NodeId	Role
Node A	123000111	Data Receiver (DR)
Node B	123000999	Data Provider (DP)

Table 3: Nomenclature for all subsequent explanations and examples showing the names, NodeIds and roles for each replication node.

4.2.1 Endpoint IDs

Names for source and destination endpoints in DTN are denoted using the URI specification (see [13]). An example for an endpoint is *dtn://some-host/some-application* addressing the application *some-application* on host *some-host*.

The basic application path for the DEM DTN proxy service is */dem*. All other features of the proxy which need to be addressable (ether to identify a source or destination endpoint) are subsequent paths of the basic path.

Analogous to the broadcast behavior for TCP/IP networks DTN supports any- and multicasting of bundles. Since the scope of a DCI may either be *ANNOUNCE* or *REPLY* a DEM DTN proxy service must provide a DTN endpoint ID (EID) for both bundles. */dem/dci/announce* is used as a destination EID for announcements and */dem/dci/reply* for the reply.

When addressing single DEM instances DEM NodeIds are used. The path for the DTN application can be constructed by the following: */dem/«NodeId»*. This EID is unique for each individual DEM instance and is used for incoming and outgoing bundles representing the TCP sockets during the lifecycle of a TMAN connection.

In addition to unique application paths each DTN host must have a unique name. This name does not necesserily have to correspond to the machine name provided by the operating system. Instead we use a name related to the military domain.

A *Call Sign* is a time limited and/or task assigned covert name used in tactical radio communications. These call signs are assigned to roles within the military hierarchy without providing any additional information about the assignee. Thus these names can be used in wiretapped environments as long as the mapping between call signs and roles remain secret. E.g. *AAT5F* migth be assigned to the role *S6 of the 1st Battalion* (Commander of the Communication Staff Section of the 1st Batallion). Within the given time and communication group calls signs are assumed to be unique.

Table 4 shows all URIs used during the lifecycle of a proxied DEM connection. Introducing a *«SessionId»* is required, because more than one connection between two DEM nodes might exist at the same time. Even though TMAN business rules allow only one connection it should be up to the DEM nodes to execute the rules. The DTN DEM proxy behaves completely transparant. A Universally Unique Identifier (UUID) is used for the *«SessionId»* (see [14]).

URI	usage	Description
dtn://dem/dci/announce	DCI	DCI announcement
dtn://dem/dci/reply	DCI	DCI reply
dtn://«Call-Sign»/dem/«NodeId»	T_OPENRQ	setting up a connection
dtn://«Call-Sign»/dem/«SRC-NodeId»/«DST-NodeId»/«SessionId»	T_DATA, T_ABRT	transmitting data and ending the connection

Table 4: URIs used for bundle source and destination addresses during the lifecycle of a DEM operation.

4.2.2 Discovery

Discovery of other DEM nodes sharing the same IP subnet is accomplished by broadcasting *DCI* data. Every DEM listens on UDP port 13152 for incoming DCI announcements. Assuming correct DCI data, the receiving DEM replies with its own DCI utilizing UDP unicast (refer to section 2.3 for details). The DCI received is stored locally, whereas the DEM NodeId and its corresponding IP address and TCP port tupel are required for establishing a DEM connection.

The DTN DEM proxy service needs to perform similar tasks (see Figure 8 for the detailed sequence diagram). It listens on UDP port 13152 for incoming DCI announcements. Upon reception of DCI announcements the tupel «NodeId», «IP-address:TCP-port» is stored. The DCI itself gets forwarded to the DTN using the DCI ANNOUNCE address *dtn://dem/dci/announce* defined in Table 4. Additionally the flag *DESTINATION_IS_SINGLETON* of the bundle header is set to *false*. This makes the bundle to be accepted by any DTN node that is used by a DTN DEM proxy. The *sender* property of the bundle header is set to the appropriate URI for the sending DEM instance, e.g. *dtn://NodeA/dem/123000999*.

When the bundle is delivered to other DTN DEM proxies they also perform a mapping between the sender's NodeId and its connection data. But instead of an IP and port tupel the *sender* property (URI of the sender) of the bundle is used.

The next step is to modify the content of the DCI received since it contains a non-reachable IP:port tupel. The Data Receiver proxy (acts like a real DEM having the role of a Data Receiver) replaces the IP:port tupel with its own interface data and broadcasts the modified DCI.

Every DEM receiving a DCI announce replies with its own DCI using UDP unicast. The DTN DEM Proxy cannnot track which reply is related to which announce. It simply takes the DCI reply and sends it to the global DTN DCI reply address *dtn://dem/dci/reply*. Again, after successful delivery of the bundle, the DCI reply is modified to meet the local IP and port settings of the proxy and then gets broadcasted locally. Unicasting the reply to the originator of the discovery process is not possible. Without the use of the proxy, a DEM instance can simply reply to the sender's IP address. Using the proxy, this correlation is lost.

After the discovery process is finished each DTN DEM proxy knows about the connection information of each DEM instance. This may either resolve to a (local) IP:port tupel or a (remote) DTN URI. See Table 5 for the address entries of the DTN DEM proxy address resolver.

Proxy Instance	Node A	Node B
Proxy A	IP:port (A)	dtn://hostB/dem/NodeB
Proxy B	dtn://hostA/dem/NodeA	IP:port (B)

Table 5: Address entries for the DTN DEM proxy resolver. Each NodeId can either be resolved to an IP:port tupel or a DTN URI.

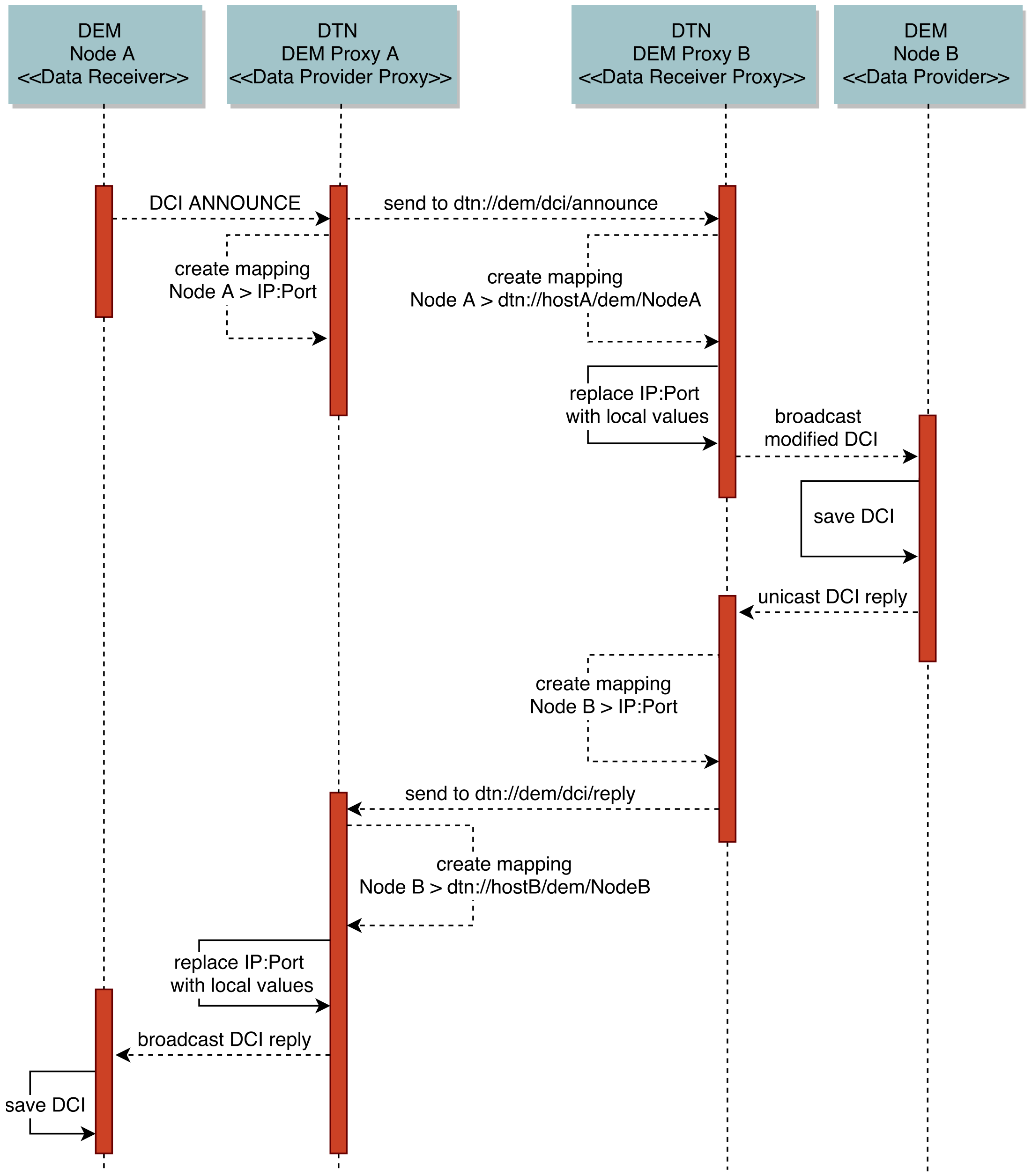

Figure 8: The sequence of operations running a DEM discovery process over the DTN DEM proxy.

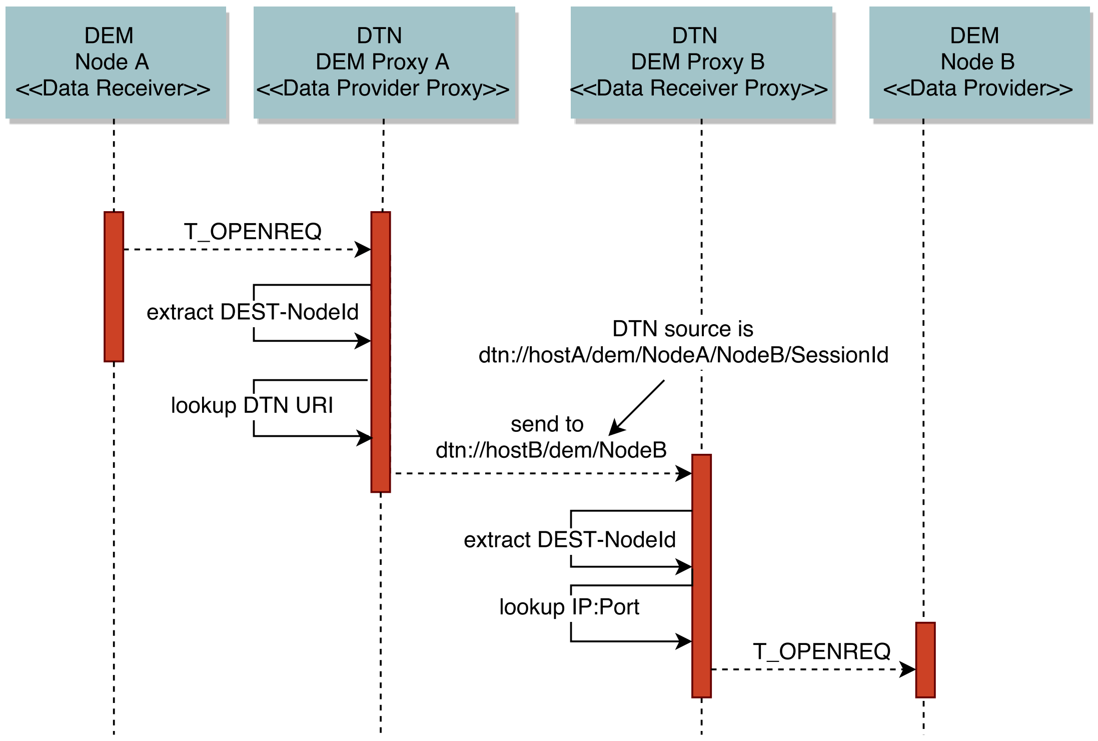

Figure 9: The sequence of operations required to establish a connection between two replication nodes.

4.2.3 Connection Setup

The preceding discovery is essential for a successful setup process since it heavily relies on the look-up tables populated during discovery. Without discovery, no mapping between «IP:port» and «DTN-URI» and vice versa exists. Figure 9 shows the necessary steps to establish a connection between two replication nodes.

Data Provider Proxy

DTN DEM proxy instances listen on configurable IP addresses and TCP ports for incoming connections from other replication nodes. Once a TCP connection is established, the initializing node (holding the role of a *Data Receiver*) sends a *T_OPENRQ* to the *Data Provider Proxy*. This message contains data about the source NodeId and the destination NodeId (see table 1 for details about the grammar). To stick with the already introduced nomenclature (see table 3), we assume *NodeA* to be willing to communicate with *NodeB*. The *T_OPENRQ* sent looks like this: `00000022T1|123000111|123000999`. The *Data Provider Proxy* extracts the destination NodeId and tries to look-up a DTN URI using the resolver component. If no URI can be found the connection is terminated. Otherwise, the *T_OPENRQ* is forwarded to the DTN URI which has been collected during the discovery process. This URI *dtn://hostB/dem/NodeB* is assigned to the "listener" of the (remote) *Data Receiver Proxy*. Any bundles which are delivered to this address are interpreted as *T_OPENRQ* messages.

To ensure the bundle gets delivered to a singleton endpoint, the header flag DESTI-

NATION_IS_SINGLETON must be *true*. The source address of the bundle is set to a "virtual path" which identifies the unique connection between two nodes. This path */dem/123000111/123000999/550e8400-e29b-11d4-a716-446655440000* (address without the scheme and host, including a random UUID) is called a *peer channel* and will be used by both parties (peers) as destination for all subsequent messages.

Data Receiver Proxy

The *Data Receiver Proxy* follows similar steps to setup a connection with *NodeB*. Triggered by the delivery of a bundle to the address *dtn://hostB/dem/123000999* the proxy extracts the destination NodeId of the contained *T_OPENRQ*. It subsequently resolves the IP:port tupel of NodeB using the NodeId, creates a TCP connection and delivers the *T_OPENRQ* to its final destination.

4.2.4 Data Messages

After establishing a connection using the *T_OPENRQ* message, the TMAN layer of DEM instances use *T_DATA* messages to encapsulate the payload of the *Data Manager* layer. The DTN DEM Proxy does not interpret the content of these messages which are just forwarded to the DEM instances. To make sure the messages are delivered to the correct DEM instance, a *peer path* was created during setting up the connection. Both proxy instances use this path (*/dem/123000111/123000999/550e8400-e29b-11d4-a716-446655440000*) in combination with their unique host names *hostA* and *hostB* to construct source and destination addresses for the bundles.

4.2.5 Connection Shutdown

Shutting down a TMAN connection is simply done by closing the underlying TCP connection. Depending on the reason for the shutdown an appropriate *T_ABRT* message will be sent.

When using the DTN DEM proxy there is only a "virtual connection" between the two DEM instances involved. Hence, shutting down the TCP connection to the proxy on either side does not affect the TCP connection on the other. Since this problem does not arise without the use of a proxy a new message must be introduced to handle the connection shutdown circumstances.

As soon as the socket to the proxy gets closed, the proxy sends a *CLOSE_SOCKET* command to its peer. The peer also shuts down the TCP connection to the DEM instance. Figure 10 shows the sequence of operations for this case.

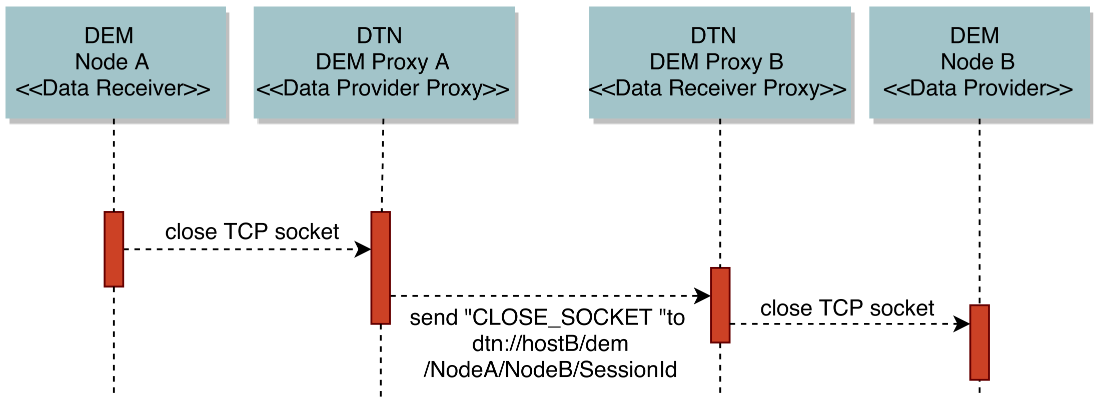

Figure 10: The sequence of operations used to shutdown a connection between two replication nodes when one side closes the TCP connection to the proxy.

4.3 Implementation

4.3.1 Design Patterns

It is the nature of the DEM protocol and especially delay-tolerant networks to be asynchronous. No assumption about timings can be made. The architecture of the DEM DTN proxy follows the asynchronous messaging design patterns described in [15]. It is event-driven and relies on a messaging subsystem. On one hand, this guarantees reliable and in-order delivery of messages exchanged between the components of the proxy. On the other hand, the principle of *loose coupling - high cohesion* can be achieved. Figure 11 shows the high level architecture of the DEM DTN proxy.

Of course, the principles of agile software development, especially S.O.L.I.D (see [16]) were also taken into account.

4.3.2 *vert.x* library

In the synchronous world, thread management itself is a difficult problem and often leads to hard-to-find bugs. Reducing the thread of execution to a single one frees developers from all *locks*, *mutexes* and other concurrency concepts to control access to memory areas. *vert.x* is a library for the *Java Virtual Machine* which allows developers to use asynchronous, event-driven concepts. It is inspired by JavaScript and the node.js runtime. The primary stage of execution is the *event loop*. Potentially blocking method calls are offloaded to background threads which will use events or callbacks when the work is done.

vert.x also makes use of the *Actor Model* (see [17]), whereas the *Actors* are called *verticles* in the *vert.x* world. The *vert.x* system is used to control the lifecycle (deploy and undeploy) the verticles. In contrast to other actor based systems (like *akka* [18]) there is no out-of-the-box actor supervision, where the supervisor is responsible for handling unexpected errors (conditional restarts if an actor stops or crashes).

verticles may consume and publish specific messages from and to the event-bus. Instead of publishing messages (the message will be delivered to every verticle registering a consumer for an given address) verticles can also send messages. This allows a point-to-point communication between verticles and also enables a *request-response* behavior.

The DEM DTN proxy is composed of multiple verticle instances which are deployed (started) by a single parent class. Figure 11 shows the high level architecture of the DEM DTN proxy.

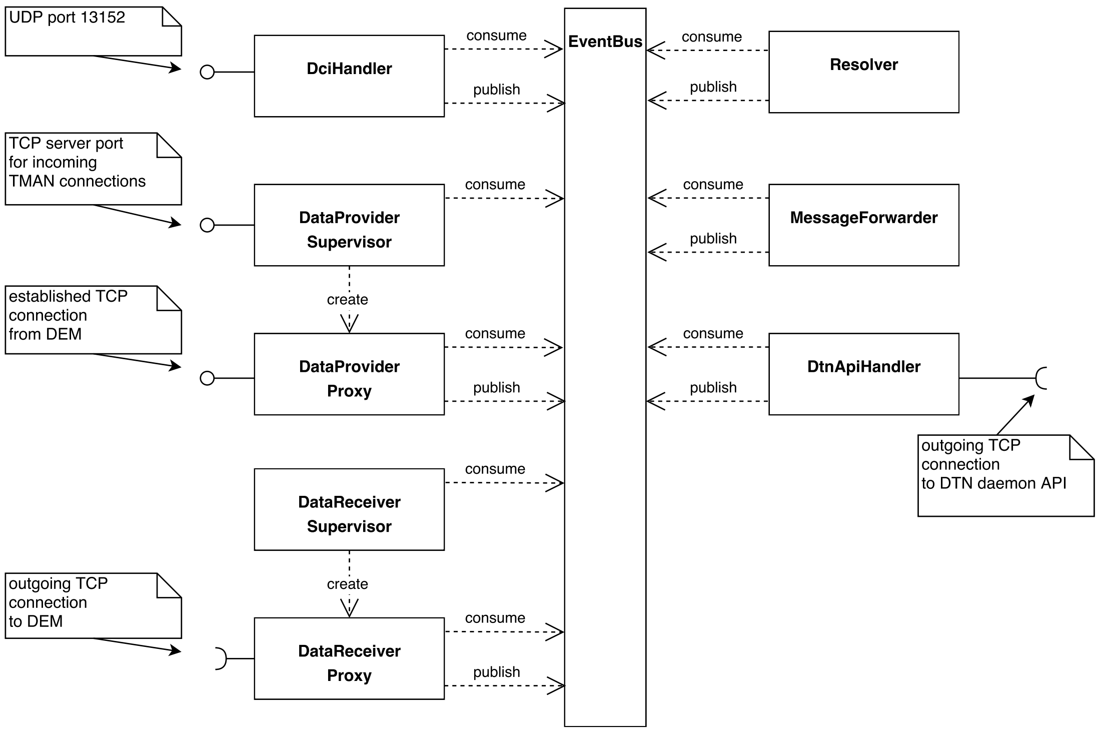

Figure 11: The high level architecture of the DEM DTN proxy service.

4.3.3 Addressing verticles on the event-bus

Verticles can register themselves on the event-bus using any kind of address - as long as the address is of type *String*. The address concept introduced for the DEM DTN proxy is based on URIs. There are two schemas, one for consuming/publishing a `command` and the other one for doing the same with an `event`.

Command names are composed of an object name (*which* object is affected) and a present tense verb to embody *what* should be done with the object. To treat the content of a message body as DCI and to broadcast it to the local network, the command `command://dci/announce` must be used.

Event names also include an object name and a past-tense verb to express *what* has happened to the object. For example if a DCI announcement was received, the corresponding event name would be `event://dci/announced`.

The objects themselves are included in the message body. If any addition data must be passed between publisher and consumer, it can be added to the message header (*key-value* pairs).

A single class concentrates the addresses and provides a constant for every address to avoid typos.

4.3.4 MessageForwarder

Following the *separation of concerns* pattern, each class should have a single responsibility. While some classes are part of the DEM domain, others are located in the DTN domain. To avoid direct knowledge and ensure loose coupling, the *MessageForwarder* class (based on the pattern of a *Message Router*, see [15] p. 78) routes messages from one domain to the other. It also translates these messages (see [15] p. 85) between the two domains (DEM and DTN).

Towards the DTN domain the *MessageForwarder* publishes `command://bundle/send` messages, while it consumes `event://bundle/received`. Sending bundles means to encode the payload using the *Base64* algorithm and to use at least a *bundle destination*. If no *bundle source* is provided to the *MessageForwarder*, the DTN daemon will set it to the default source (a random address created when the deamon starts). When a bundle is received, the *MessageForwarder* compares the destination address of the bundle with the predefined DCI addresses for announcements and replys. In case of a match an appropriate message is published on the event-bus. Otherwise the payload gets published to the destination address on the event-bus assuming an existing consumer. A *Resolver* is used to map a NodeId either to the address spaces of the DEM (IP:port) or to the one of the DTN (URI).

4.3.5 DciHandler

Running the discovery process is essential for the subsequent handling of messages. The *Dci-Handler* listens on the UDP port 13152 for incoming DCI messages. Upon reception, the local DCI is published on the event bus using `event://dci/announced` or `event://dci/received` where it gets consumed by the *MessageForwarder* and forwarded to the ANNOUNCE or REPLY URI defined for the DTN (see 4.2.1). In addition to this, the message is consumed by the *Resolver* to create a mapping between the local IP:port tupel and the NodeId extracted from the DCI.

Whenever a DCI is delivered by the DTN to the proxy, is gets published on the event-bus using `command://dci/announce` (or `command://dci/announce`). The *DCIHandler* consumes these messages and broadcasts them on the local IP interface. To avoid creating infinite loops the *DCIHandler* only accepts messages sent by other hosts and drops the ones emitted by itself.

4.3.6 DataProviderSupervisor

Normal DEM instances listen on a configurable TCP port for incoming connections. So does the *DataProviderSupervisor*. Whenever a new connection is established, the supervisor creates a new instance of a *DataProviderProxy* and delegates control over the socket to it. The *DataProviderSupervisor* itself is not involved in any communication details between the DEM node and the *DataProviderProxy* and does neither consume nor publish messages from/to the event bus.

4.3.7 DataProviderProxy

The *DataProviderProxy* manages the incoming communication with a DEM node. It gets started with a *PreT1* behavior where it waits for the initializing *T_OPENRQ* (T1) message. After the successful extraction of the source and destination NodeIds the *T_OPENRQ* is published on the event-bus using `command://tman/send` where it gets forwarded by the *MessageForwarder*. The internal behavior of the *DataProviderProxy* now switches to the *initialized* state. The *peer channel address* (see 4.2.1) is used to consume messages from the event-bus. To send messages, the `command://tman/send` in conjunction with two message header keys (source and destination) is used. No additional analysis of incoming TMAN PDUs is done now, they just get published on the event-bus.

Shutting down a *DataProviderProxy* instance (and undeploying the verticle) can be triggered by two ways. Either by closing the TCP conection between the DEM node and the *DataProviderProxy* which creates a `command://socket/close` on the event-bus. Or by consuming a `event://socket/closed` message initiated by closing the TCP connection to a remote *DataReceiverProxy* instance (see 4.2.5).

4.3.8 DataReceiverSupervisor

Being the counterpart of the *DataProviderSupervisor*, the *DataReceiverSupervisor* listens on the event-bus to the delivery of messages intended to create a new outgoing TCP connection to a DEM node (see 4.2.1). It extracts the destination NodeId contained in the *T_OPENRQ* and queries the *Resolver* for the corresponding IP:port tupel. Using this information it creates a new instance of a *DataReceiverProxy* and delegates further actions to this verticle.

4.3.9 DataReceiverProxy

When instanciated, the *DataReceiverProxy* tries to establish a TCP connection using the IP:port tupel provided on creation. The second constructor parameter is the *T_OPENRQ* which has been received by the *DataReceiverSupervisor*. As soon as the TCP connection to the destination DEM node is created successfully, the *DataReceiverProxy* transmits the *T_OPENRQ*.

If the process succeeds, the *DataReceiverProxy* registers itself for the peer channel address on the event-bus to allow subsequent communication. Similar to the *DataProviderProxy*, this makes sure that there is a virtual connection between the source and the destination DEM node.

Considering the case where establishing the connection fails or the DEM node closes the connection, a *CLOSE_SOCKET* message ist sent to the peer channel address. If the DEM node transmitted a *T_ABRT* message, it will be forwarded to the peer prior to the *CLOSE_SOCKET* message.

4.3.10 DtnApiHandler

Communication with the *IBR-DTN API* is accomplished by sending and receiving API commands and events over TCP connection to the API daemon. The API provides at least seven *protocols*, each of which is designed to manage a different aspect of the DTN daemon. The DEM DTN proxy empowers the `protocol extended`. The *request-response* pattern is used to send commands to, or query data from, the daemon. In both cases the daemon answers using a *«CODE» «MESSAGE»* format. The codes are similar to the http protocol status codes. For example, `200 OK` is returned by the daemon to indicate that the execution of the last command has been successful. Since the DTN daemon is also event driven it indicates the presence of a new bundle by presenting a `602 NOTIFY BUNDLE` message. These messages may be sent at any time, even after the invocation of a command and before the receipt of the corresponding response.

The *DtnApiHandler* encapsules the handling of the DTN API. It consumes bundles from the event-bus wich were sent to the `command://bundle/send` address. These bundles get serialized and transmitted to the DTN daemon. When the *DtnApiHandler* gets notified by the DTN daemon, it parses and de-serializes the chunks of the bundle. After completion, the bundle gets published on the event-bus using the `event://bundle/received` address.

4.4 Verification

To prove the implemented functions of the DEM DTN proxy, both the discovery and the connection phase need to be verified. Since the data exchanged is ordinary text one could use simple tools like *telnet* and *netcat* (nc) to act instead of real DEM nodes. Both tools are available for all major operating systems or can easily be installed. But simulating and verifying more sophisticated or time triggered message sequences (like the T_CHKCON message) will not work. Staying in the world of the asynchronous programming paradigm, this work uses *Javascript* together with the *node.js* runtime to implement the functions needed to verify the DEM DTN proxy. Each step is denoted by using assertions borrowed from unit testing frameworks.

4.4.1 Environment

The environment to verify the DEM DTN proxy can easily be put together using a virtualized system. The DEM DTN proxy instances coexist with the DTN daemons on the same machines. To comply with real scenarios the commands to start discovery and establish a TMAN connection are invoked from a machine different from the proxy machines.

The DTN daemons only use TCP convergence layers and are connected to an isolated IP subnet. Discovery of DTN nodes is done by sending discovery beacons every five seconds.

All dynamic routing mechanisms are disabled, bundles being only exchanged between directly connected nodes. The default TCP port 4550 is kept for the DTN daemon API. Figure 12 shows the test setup environment.

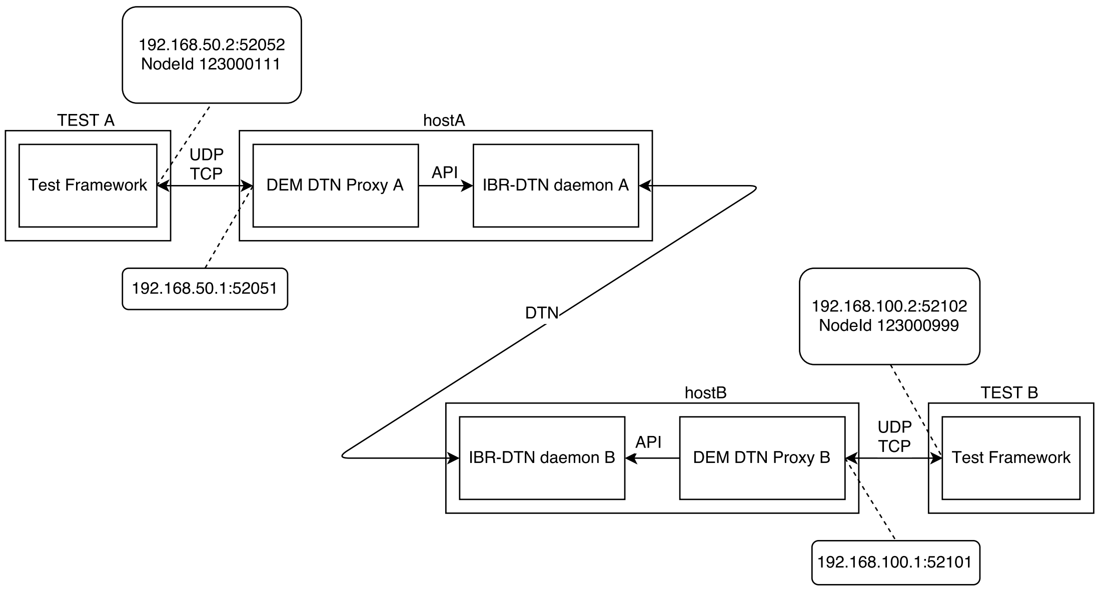

Figure 12: Proxies, DTN daemons and test tools for verifying the functionality of the DEM DTN proxy implementation.

4.4.2 Discovery

In order to verify the discovery process we prepared two DCI XML files, one for each Test node. *TEST A* will send a DCI ANNOUNCE.XML and *TEST B* will answer by using a REPLY.XML file. Each file contains all mandatory properties for a DCI (see 2.3). The NodeId of *TEST A* is 123000111, and the one of *TEST B* is 123000999.

When broadcasting the DCI from *TEST A*, the following assertions must be true:

- *TEST B* should receive a DCI on UDP port 13152.
- The scope of the DCI should be *ANNOUNCE*.
- The NodeId should be 123000111.
- The DCI should contain the IP of *hostB* (192.168.100.1)
- the DCI should contain the TCP port of *hostB* (52101)

To complete the discovery process, *TEST B* answers with its DCI. The verification is finished successfully if the following are true:

- *TEST A* should receive a DCI on UDP port 13152.

- The scope of the DCI should be *REPLY*.
- The NodeId should be 123000999.
- The DCI should contain the IP of *hostA* (192.168.50.1).
- The DCI should contain the TCP port of *hostA* (52051).

4.4.3 Connection Setup

To verify the successful establishment of a communication channel between *TEST A* and *TEST B* we need to exchange messages in both directions.

If *TEST A* opens a TCP connection to 192.168.50.1:52051 (*hostA*) and transmits a *T_OPENRQ*, the assertions listed below must evaluate to true:

- A TCP connection should be opened from 192.168.100.1 (*hostB*) to *TEST B*.
- *TEST B* should receive `00000022T1|123000111|123000999`.

Real-life DEM implementations will send the available OIG list after the TMAN connection is established. To be compliant with this rule *TEST B* sends a TMAN message containing an appropriate Data Manager message.

TEST A asserts that:

- It should receive a T_DATA message.

4.4.4 Connection Shutdown

Terminating the connection can be initiated either by *TEST A* or *TEST B*.

In the case *TEST A* closes the TCP connection to *hostA* we assert that

- The TCP connection from *hostB* to *TEST B* should be closed.

Otherwise, if *TEST B* closes the TCP connection because of a simulated error condition we assert the following to be true:

- *TEST A* should receive a T_ABRT message containing an error code.
- The TCP connection from *hostA* to *TEST A* should be closed.

4.5 Test with existing DEMs

To make sure the primary objective was achieved, the DEM DTN proxy was also tested with an existing DEM implementation provided by *syncpoint GmbH*. Their DEM passed a very large number of System-Level Tests (SLTs) stage 1 and stage 2 (SLT1 for the communication protocols, and SLT2 for database replication). These tests are part of the MIP Test Reference System (MTRS) which is maintained by the *Fraunhofer Institute for Communication, Information Processing and Ergonomics*. See [19] for a brief introduction to MTRS.

After replacing the test hosts *TEST A* and *TEST B* with the syncpoint DEM implementations all verification steps were repeated and finished successfully. This substantiates the objective of this work to create a fully transparent, application-aware DEM proxy.

4.6 Results

The implementation of the DEM DTN proxy presented in this work has passed all verification steps defined in 4.4. The accomplishment of the primary objective was proven by testing the phases of the DEM communication by using a simple test framework. Additionally, an existing DEM implementation was employed to make sure no obvious flaws in the design or implementation were left over.

Since the DTN environment has been limited to only use almost fail-save features, no prediction about the behavior of the proxy under real-life circumstances can be made. Examining the performance of the proxy "in the wild" can be part of future work.

5 Summary

Delay- and Disruption-Tolerant Networks allow clients to communicate even with no reliable end-to-end path available. This is made possible by introducing a store-carry-forward mechanism for *bundles*. The author of [12] created an event-driven implementation (IBR-DTN) for the existing DTN RFCs 5050 and 6257.

Employing the *IBR-DTN* extended API this work shows the architecture and implementation of a transparent, application-aware proxy service for the MIP Data Exchange Mechanism. The TMAN protocol is a quite simple, text-based protocol. Of course, all phases of the DEM connection protocol are supported. During discovery relevant parts of the DCI must be replaced by the proxy to pretend the existence of a real DEM.

The proxy itself is implemented in Java and makes use of an asynchronous, node.js inspired library called *vert.x*. This supports the asynchronous nature of the TMAN messages, which are not timing-related in any way. Each TCP connection established to or from the proxy is handled by a vert.x verticle. Mapping TCP socket addresses to DTN URIs was a challenge during development and is realized using a Resolver verticle.

To verify the design and implementation a simple JavaScript/node.js test framework was used. This framework allows end-to-end tests. Taking the verification one step further the proxy was also tested with an existing DEM which proves the correct implementation.

5.1 Future work

The current design employs a single *MessageForwarder* that routes messages between the DEM and the DTN domain. There also exists only one single *ApiHandler*. While this migth not be a performance bottleneck an effort is needed to register and handle the appropriate message consumers. An even simpler design without the need of a centralized router seems to be promising. After the initial setup each peer channel will be handled by a dedicated message pipe. Also, resolving addresses between IP:port tupels and DTN URIs will only be needed during the connection setup phase.

TCP ensures the correct order of packets. When transmitting data over a DTN, the order of the bundles cannot be guaranteed anymore. Since the DEM protocol is just a supplier for the JC3IEDM data model, messages must be delivered in order. IBR-DTN provides a dedicated protocol designed for streaming applications. This protocol already has an ordering mechanism for bundles.

The current implementation of the DEM DTN proxy does not evaluate administrative messages of the DTN. These messages are comparable to ICMP and indicate the state of delivery for bundles (e.g. delivered, forwarded, discarded). Since the DEM must ensure reliable transmission, TCP connections to DEM nodes shall be closed if a bundle was not delivered within its lifetime of if the bundle was discarded.

No attention has been given to optimization issues. The DEM node holding the DR role sends T_CHKCON message on a regular schedule to keep the TCP connection alive. It doesn't make any sense to transmit these messages over the DTN, because the proxy on the DP side can do this on its own.

This work made some assumptions on the DTN environment to avoid interfering parameters. To prove its functionality, the DEM DTN proxy should be tested under real-life conditions, using multiple DEM nodes and a more challenging network environment.

One of the target environments for the DEM DTN proxy is the use of VHF radios in tactical networks. These radios often allow data communication using a serial communication port. While some of the competitors of IBR-DTN already have convergence layers for serial communication IBR-DTN lacks this feature.

5.2 Related work

Scott [20] focuses on SMTP and http. For http he extends the existing *World Wide Web Offline Explorer* (WWWOFFLE) to make use of DTN. He splits up the WWWOFFLE into a client and a server side. The client remains in the challenging network and uses DTN to communicate with the server. The server is assumed to be well connected and processes the requests received from the client. To minimize the numbers and the size of the bundles, only a subset of html features is supported. E.g. CSS processing is omitted.

Dalecki et al. [21] also use DTN for their work about *Military Disruption Tolerant Networks* (MIDNet). They discuss three protocols for tactical use, although these protocols (http and SMTP/POP3) are not restricted to tactical applications. Even the *Blue Force Tracking* functionality is assumed to work over http. They use a proxy server for each protocol to intercept the application data and put it into DTN bundles. As expected, minor performance degradation were observed. While SMTP and POP3 are asynchronous by nature http is used in a more synchronous way (a user waiting for the response). In order to enhance user experience, web data caches were used. In contrast to Scott, their solution supports the full feature-set of html.

All protocols examined in the work mentioned above are client-server protocols. At least the DTN node hosting the server is assumed to be stationary because is acts as a bridgehead to well-connected networks. The address of the server node also remains static. In contrast to this the DEM protocol discussed in this paper is a peer-to-peer protocol where discovery is part of the protocol definition. Additionally all nodes participating are allowed to be mobile and may change their location rapidly.

In [22] a DTN proxy for the *Jabber* instant text messaging (chat) protocol is presented. To optimize the network traffic required for the protocol they introduced a multicast extension for the *DTN2* reference implementation [11]. This way the chat server can publish data only once but DTN delivers the message to all clients that subscribed to a DTN multicast endpoint id. This is a very interesting approach and may be taken into account for the dissemination of DEM messages to multiple subscribers. In contrast to the proxy server presented in this work the prototype for the Jabber proxy does not support changes at runtime and thus clients cannot join multicast groups dynamically.

Acronyms

BER	bit-error rate
C2IS	Command and Control Information System
CWIX	Coalition Warrior Interoperability eXploration, eXperimentation, eXamination, eXercise
DCI	DEM Connection Information
DEM	Data Exchange Mechanism
DMAN	Data Manager
DP	Data Provider
DR	Data Receiver
DTN	Delay- and Disruption-Tolerant Network
EBNF	extended Backus–Naur form
EID	endpoint ID
FFI/MTF	Friendly Force Information/Message Text Format
JC3IEDM	Joint Command Control and Consultation Information Exchange Data Model
LTE	Long Term Evolution
MANET	Mobile Ad-Hoc Network
MIP	Multilateral Interoperability Programme
MTRS	MIP Test Reference System
NATO	North Atlantic Treaty Organization
NFFI	NATO Friendly Force Information
OIG	Organisational Information Group
PDU	Protocol Data Unit
SLT	System-Level Test

TMAN	Transport Manager
UUID	Universally Unique Identifier
VHF	Very High Frequency
WIMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

Bibliography

[1] E. Golan, A. Kraśniewski, J. Romanik, P. Skarżyński, and R. Urban, "Experimental performance evaluation of the narrowband VHF tactical IP radio in a real environment," in *Military Communications and Information Systems Conference (MCC), 2013*. IEEE, 2013, pp. 1–5.

[2] I. Chlamtac, M. Conti, and J. J.-N. Liu, "Mobile ad hoc networking: imperatives and challenges," *Ad hoc networks*, vol. 1, no. 1, pp. 13–64, 2003.

[3] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 27–34.

[4] R. Durst, K. Scott, E. Travis, and H. Weiss, "Interplanetary Internet (IPN): Architectural Definition V. Cerf Worldcom/Jet Propulsion Laboratory S. Burleigh, A. Hooke, L. Torgerson NASA/Jet Propulsion Laboratory," 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.7347&rep=rep1&type=pdf

[5] A. F. Lindgren, A. Doria, and E. Davies, "Probabilistic Routing Protocol for Intermittently Connected Networks," Internet Requests for Comments, RFC Editor, RFC 6693, August 2012. [Online]. Available: https://tools.ietf.org/html/rfc6693

[6] Multilateral Interoperability Programme, *MIP Technical Interface Design Plan, Edition 3.1.2 Annex B*, 2012. [Online]. Available: https://www.mip-interop.org/Public%20Document%20Library/04-Baseline_3.1/Interface-Specification/MTIDP/MTIDP-3.1.2-AnnexB-MIP_DEM_Specification.pdf

[7] E. Larsen, "TCP in MANETs–challenges and Solutions," *FFI-Rapport-2012/01514*, 2012. [Online]. Available: http://rapporter.ffi.no/rapporter/2012/01289.pdf

[8] K. L. Scott and S. Burleigh, "Bundle Protocol Specification," Internet Requests for Comments, RFC Editor, RFC 5050, November 2007. [Online]. Available: https://tools.ietf.org/html/rfc5050

[9] S. F. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle Security Protocol Specification," Internet Requests for Comments, RFC Editor, RFC 6257, May 2011. [Online]. Available: https://tools.ietf.org/html/rfc6257

[10] F. Warthman *et al.*, "Delay-and disruption-tolerant networks (DTNs)," *A Tutorial. V3.2, Interplanetary Internet Special Interest Group*, 2015. [Online]. Available: http://ipnsig.org/wp-content/uploads/2015/09/DTN_Tutorial_v3.2.pdf

[11] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra, "Implementing Delay Tolerant Networking," Technical Report IRB-TR-04-020, Intel Research, Tech. Rep., 2004.

[12] J. Morgenroth, "IBR-DTN - Event-driven Software-Architecture for Delay- and Disruption-Tolerant Networking," phdthesis, TU Braunschweig, Carl-Friedrich-Gauß-Fakultät, Institut für Betriebssysteme und Rechnerverbund, 2015. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00061364

[13] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," Internet Requests for Comments, RFC Editor, RFC 3986, January 2005. [Online]. Available: https://tools.ietf.org/html/rfc3986

[14] P. Leach, M. Mealling, and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," Internet Requests for Comments, RFC Editor, RFC 4122, July 2005. [Online]. Available: https://tools.ietf.org/html/rfc4122

[15] G. Hohpe and B. Woolf, *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[16] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[17] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[18] Lightbend Inc. (2016) Akka: A toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM. [Online]. Available: http://akka.io

[19] M. Gerz, N. Bau, A. Vogt, and R. Vogt, "Automated Conformance Testing of C2IS," DTIC Document, Tech. Rep., 2009.

[20] K. Scott, "Disruption tolerant networking proxies for on-the-move tactical networks," in *MILCOM 2005-2005 IEEE Military Communications Conference*. IEEE, 2005, pp. 3226–3231.

[21] T. Dalecki, M. Mazur *et al.*, "Adapting standard tactical applications for a military disruption-tolerant network," in *Military Communications and Information Systems (ICMCIS), 2016 International Conference on*. IEEE, 2016, pp. 1–5.

[22] R. Metzger and M. C. Chuah, "Opportunistic information distribution in challenged networks," in *Proceedings of the third ACM workshop on Challenged networks*. ACM, 2008, pp. 97–104.