A distributed delay tolerant data transfer protocol for ISTSAT-1

Pedro Gameiro
pedro.a.gameiro@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

A cubesat is a very small, low cost, artificial satellite designed for space research purposes, very popular in the academic community. The ISTSAT-1 is the first cubesat being developed in Instituto Superior Técnico (IST), with a size of a 10x10x10cm cube respecting the 1U cubesat standard.
In space communications one finds different challenges from the ones on Earth. Links are typically unstable, of low debit, of high delay and disruptions can be frequent. This motivated the creation of Delay Tolerant (DT) protocols, an architecture designed to deal with the characteristic problems of disruptive environments. However, even with DT, the transmission of big data files can be difficult. This is specially true in the case of cubesats. Due to their Low Earth Orbit (LEO) deployment, cubesats suffer from very long disruptions periods with the ground-stations on Earth.
With this project we intend to create an enhanced solution. A distributed DT protocol capable of performing the normal DT tasks, but in a manner where a single transmission can be distributed over different devices (like ground-stations or satellites), thus increasing the number of links, reducing the disruptions time periods and increasing the much needed performance of space transmissions.

Keywords: IST, nanosat, LEO, ISTSAT-1, distributed

1. Introduction

Space research has traditionally been limited to organizations who can afford the large investments associated with it. This presents a major limitation in technological development.
A popular new trend, specially in universities and the academic community, is the development of miniaturized satellites known as cubesats[3]. These satellites offer a major opportunity for organizations with limited resources to perform space research[11]. This is not only due to their small sizes, but since they can be built with common electronic components they are of low cost. Its design was proposed in 1999 by professors Jordi Puig-Suari of California Polytechnic State University and Bob Twiggs of Stanford University.
Standard cubesats usually have a volume of 1 liter (10^3 cm cube), weight 1kg and are called 1U(one unit) cubesats. There are also some bigger models, scaled in the vertical axis, 2U(20x10x10cm) and 3U(30x10x10cm) and even as big as 6U and 12U.
ISTSAT-1[6] is the first cubesat being developed in Portugal, by Instituto Superior Técnico - University of Lisbon (IST), with a 1U size (10^3 cm cube) by students, professors and radio-amateurs. The ISTSAT-1 mission is first and foremost an educational one, joining together multiple students from different expertise and engineering programmes to foster their enthusiasm for space science and complementing their education with a challenging hands-on project.
In 2014, under unexplained circumstances, the Malaysia Airlines flight 370 plane disappeared in the middle of the ocean. This event triggered a new discussion about safety in the airline industry and a growing interest in tracking airplanes in remote areas. In November 2015, the International Telecommunication Union's World Radio communication Conference (ITU WRC-15) allocated a new frequency band to be used by the Automatic Dependent Surveillance—Broadcast (ADS-B)[9] system in earth-space direction[1]. This enabled the possibility of tracking aircraft positions from space, at a global scale. Today, this signal is only monitored through ground-based stations, with no coverage in oceanic routes and remote areas. The ADS-B system is a security feature that most planes today use, and all of them will be obliged to use by 2020. It periodically (each 0.5 seconds) emits a message with information about the plane, it's location, identification and status. ISTSAT-1[5] will be used for carrying a feasibility study of the use of cubesats for

[1] https://www.itu.int/net/pressoffice/press_releases/2015/51.aspx

receiving ADS-B signals from traveling airplanes, as illustrated in figure 1. The ISTSAT-1 spacecraft is also being developed in the context of an European Space Agency (ESA) educational programme, the Fly Your Satellite (FYS), along with 5 other european teams developing their own cubesat[2]. In this programme the ISTSAT-1 development will be assisted by experienced engineers and professionals from ESA that have developed, launched and operated several spacecrafts in their carer. This is a major opportunity that as drastically impacted this project. The FYS programme as also offered the ISTSAT-1 a launch opportunity, currently schedule for 2020.

To address the problems associated with space communications (characterized by long distances, high delay environments, frequent disruptions or disconnections), a new communication paradigm emerged, the Delay/Disruptive Tolerant Networking (DTN). The DTN specification defines an hop-to-hop architecture where link disruption is acceptable and transmissions will pause and resume as necessary. This paradigm is already being used in the National Aeronautics and Space Administration (NASA) mars rover mission[3].

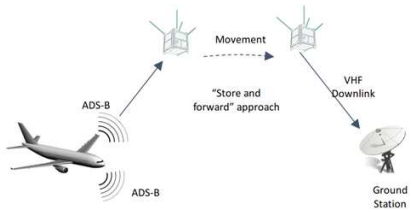

Figure 1: ADS-B System

2. Motivation

Cubesats are normally deployed in Low Earth Orbit (LEO), which is defined as 300-1500km above the earth surface, with an orbital period of approximately 90 minutes at a speed of 7.8 km/s. However, a big consequence of this, is that a particular ground-station only has a very small time-window (15-20 minutes) of line-of-sight to communicate with the satellite. Offering even more obstacles, space-link communications are very unstable, error prone, and of low debit. The challenges in space-link communications are not traffic congestion (like on earth), but long propagation delays and high bit-error rates.

The current approach to this problem is the use of DTN [13][10] protocols. These (in contrast to common, terrestrial protocols) are designed to tolerate large periods of delay (or disrupted links) and transmit at high bitrates when the link is available. This approach allows transmissions to be paused when the communication space link is unavailable, and to resume when a line-of-sight is once again available.

The purpose of this project is to tackle this difficulties in a new, more powerful way. To create a distributed protocol capable of expanding a single transmission session over several links and hosts (Ground Stations (GSs) in this particular case).

This is illustrated in figure 2, where the satellite, after starting a transmission with a given GS, leaving its line-of-sight and consequently disconnecting the link, does not need to perform a full orbit to resume the transmission. It can just continue the transmission with the next GS available in its orbit. This can result in a very significant performance enhancement, both due to the very limited available time-windows and the typical characteristics of space communications, like high delays and low throughputs.

Another interesting use case for such a solution is the transmission of data through, and within, cubesat clusters[4]. In this types of spacecraft constellations, such as the humsat project [4], transmissions can be tricky since every single node is moving on its own orbit, and also relative to earth's GSs. Due to this, a distributed approach like the one proposed by this project can be very useful. With enough cubesats in a cluster, and taking advantage of a distributed solution, it could even be possible to create a real-time service for tracking ADS-B signals.

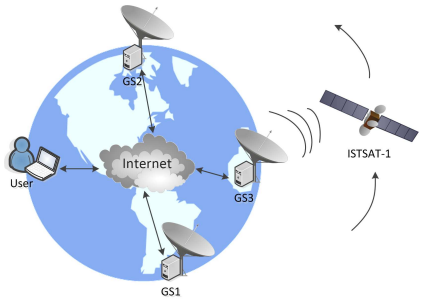

Figure 2: ISTSAT-1 Distributed transmission

3. Objectives and challenges

With this project we intend to create a Delay/Disruptive Tolerant (DT) protocol, prepared to deal with space communications related challenges that are already being dealt with in today existing DTN protocols, but one that is capable of transmitting in a fully distributed manner. This means that, not only this protocol will be able to deal with disrup-

[2] https://www.esa.int/Education/CubeSats_-_Fly_Your_Satellite/Six_new_CubeSat_missions_selected_for_next_cycle_of_Fly_Your_Satellite

[3] http://mars.nasa.gov/mer/home/

[4] https://www.humsat.org

tive prone environments, but it will also be capable of performing a single transmission through different nodes.
In a typical cubesat communication, a DT transmission occurs between a ground station and a satellite (optional with other hops). This is not enough, specially when transmitting big data files. With the typical DT protocols, this is handled by resuming the transmission (where it was left of) when the cubesat completes an orbit and the line-of-sight is restored. Note, however, the passes over the GS receiving (or transmitting) the big data are not always azimuth-favorable in consecutive orbits due to its quasi-polar characteristics[1]. Therefore, it might take a while to complete the entire transmission successfully for a given GS.
With a distributed protocol the transmission could be performed over different ground-stations, located in different geographical locations.
The proposed protocol should be bidirectional (i.e. capable of transmitting both in the uplink and downlink), capable of prioritization when dealing with several simultaneous transmissions, guarantee the correct integrity of the received data and retransmit the broken segments. These are all common features found in transport protocols.
This protocol is intended to be used over Open Systems Interconnection (OSI) layer 4 Reliable Datagram Protocol (RDP) protocol that in its turn will use Cubesat Space Protocol (CSP). CSP is a very popular, well tested, protocol for cubesat systems. It was designed for the purpose of space transmission and it is able to address different, possibly independent, modules of a given satellite. It works in both OSI layers 2 and 3 and its original implementation source code is freely available for download and changes.
During the development of this project, a scientific paper[8] was also written. It features the designed solution and some preliminary results. It was presented and published at the 15th International Symposium on Wireless Communication Systems (ISWCS) conference in 2018.

4. Protocol

The implemented Distributed Delay Tolerant Protocol (DDTP) protocol design was inspired on both the Saratoga protocol[2] and the peer-to-peer architecture[7].
In this section the term "transmission" will be used to describe a network communication between two directly connected nodes, and the term "session" to describe a DDTP, distributed, connection that spans several transmission over several nodes.
The proposed solution must outperform the current solutions, speeding up transmission times by the use of its distributed architecture.
The major purpose of the implementation is the transmission of large data payloads from and to LEO cubesats, but it is also intended to solve the transmission of data in spacecraft clusters topologies.

4.1. Messages

A total of five message are used to communicate between network nodes, these are illustrated in figure 3:

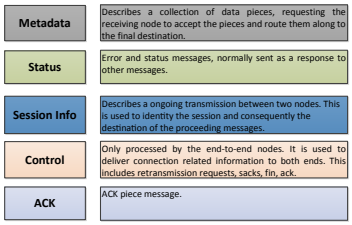

Figure 3: DDTP Message types

4.2. Algorithm

The protocol algorithm is defined by two state machines one for controlling the reception and another the transmission procedures. Moreover, one of the core rules of a session is that when a block is exchanged, the receiving node takes in the responsibility for that block, meaning it should keep forwarding it along the route that will eventually reach it final session destination.
For route determination each node looks at a table kept in memory. Each entry in the table contains metric, destination and route values. There are several possibilities for how the routing table values could be determined. The simplest one consists in static values, with a lot more complex examples being possible, like dynamically changing the table to reflect the best space-link route computed by analyzing the orbital parameters of a spacecraft orbit[12]. This is however outside the focus of this dissertation.
Data blocks then flow trough the network, in no particular order, until they arrive at their destination. If a data block does not arrive after a long time (e.g. node crashed or a particular link has been offline for a long time) the receiving node sends a **control** message (figure 3) with a SNACK flag for the block in question.
One of the protocol requirements is that it should be usable in low resource embedded systems, so some compromises need to be made. Instead of identifying each data block that is going to be sent through a cryptographic-safe hash, like in the case of a typical peer-to-peer protocol, a sequence number along with the transmission session ID is sent.

Each active node in the network starts in the standby state. This state is where the protocol remains when it has no work to do, more specifically when it doesn't have anything to send or receive. There are two ways to leave this state, either a network communication is received from another node, moving therefore to the reception state machine (figure 5), or a request for a new transmission of data is received from the upper layers of software, moving the protocol to the sender state machine (figure 4). This process also moves the receiver into the **pre recv session info** state and the sender to the **pre send session info** state.

In every exchange between two peers the **Session info** message (see figure 3) is always the first one sent. This message describes the session in question, its identification (the source address, session id pair) and priority. The transmission and reception of this message moves the receiving peer from the **pre recv session info** to the **pre recv** state, as illustrated in figure 5.

If the sender intends to send data blocks, it will move to the **pre sending** state, but if a control message is to be sent, it will move to the **send control** state, as illustrated in figure 4.

In the case of a **control** message, the receiver moves to the **control** state. At this point the receiver can send a message either accepting or rejecting the control **message**. If the message is reject, its up to the sender to find another route for delivering it to it's destination. If however it is accepted, the receiver will store the message and forward to its destination. This moves both machine states to the end state. **Control** messages are only processed by the end-to-end transmission peers, every other peer will not look into its contents. The only exception to this rule is control messages with a retransmission flag. These may trigger a data block transmission in a intermediary peer if the peer contains the data block in storage, either in alive or zombie state.

If a **metadata** is received, however, the receiver move to the **pre recv 2** state, will parse it and retrieve the information about the data blocks that the sender intends to send. The sender on its hand will move to the **pre sending 2** state. The receiver will respond with a **status** message reporting if the transmission is accepted or not. Reasons for rejecting include lack of storage space, no route available for the session and power saving modes.

If the transmission is accept, however, the sender move to the **sending** state and starts sending the data blocks described in the **metadata** message until completion or the link drops, while the receiver move to the **recv** state and receives the mentioned data blocks.

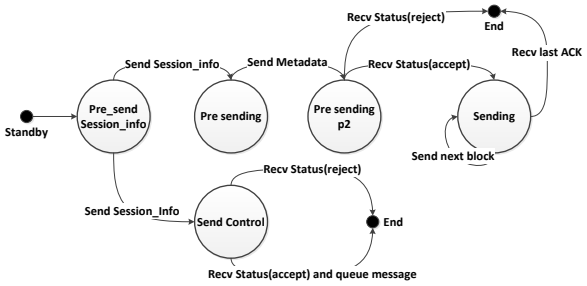

Figure 4: DDTP sender process state machine

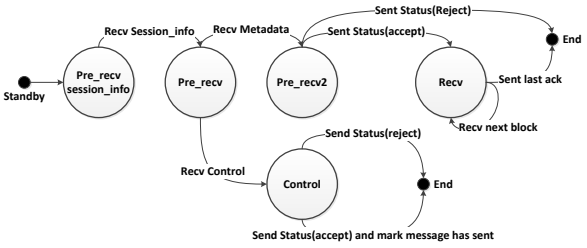

Figure 5: DDTP receiver process state machine

5. Application Scenarios operations

Taking in consideration the solution proposed in section 4, the operational behavior of the developed protocol will now be explained in each scenario. This is a theoretical description that will focus on the more technical aspects, experimental results will presented in section 9.

5.1. One-to-one transmissions

Even though the first scenario is the most simple one, and it is already solved by the current solutions, it is still very relevant. This is because it allows one to test the protocol algorithm in a simple case first, before going to the more complex scenarios, and because it allow one to compare the proposed solution performance with the current solutions in the same context.

In figure 6 a simplified Message Sequence Chart (MSC) of a transmission between two, directly connected, nodes is depicted. In this example, node **A** initiates the transmission by sending a **session info** and **metadata** messages. The **session info** describes the session itself. It contains the addresses, priority and session id. The **metadata** message describes the data blocks that will be sent in this particular transmission.

The receiver (node **B**) after receiving these messages will have all the information it needs to "decide" if it is able to accept and forward along the data blocks.

Since in this particular example node **B** is the final destination, it will sent a **status** message informing node **A** that it accepts the transmission. After this node **A** starts sending data blocks.

This process is very similar to the one described in section 8, which is to be expected since to be expected since their both solving the same scenario. The DDTP exchange however requires a higher network overhead since more metadata is sent. This may be decremental in this particular scenario, but will result in a much higher total throughput in the new scenarios.

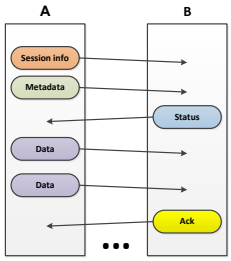

Figure 6: MSC One-to-one transmission

5.2. One-to-multi transmissions

In figure 7 an example of this scenario is displayed. The major difference in this scenario is the usage of multicast messages. This means that one cannot rely on the receivers acquiring all the messages (there is not re-transmissions at the transport layer). This means that the data messages need to be sent together with the information about the DDTP session. This is displayed in the example figure, over the RDP layers, the necessary fields are inserted in the DATA message. There is offcourse no way to ensure that all the data blocks arrive at all the receivers correctly, but for this the retransmission message explained in section 4.1 will be use by the receiver to request a normal, unicast, retransmission of the mission data blocks.

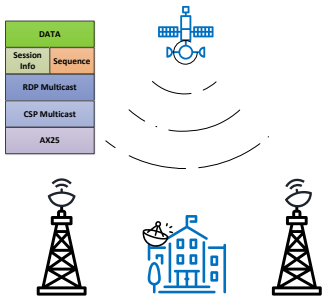

Figure 7: One-to-multi transmission

5.3. Distributed transmissions

In this section, while using the concepts introduced in the previous scenarios, a distributed transmission will be explained. In figure 8 an example is illustrated, where each black arrow corresponds to a one-to-one transmission (explained in section 5.1) and each red arrow represents a link drop. The example is divided in three steps, A, B, and C that correspond to three transmission opportunities in an orbit. The example corresponds to a spacecraft (ISTSAT-1 in the figure) downlinking to earth a data payload through multiple GSs. This is the major use case for which this protocol was developed.

When the spacecraft (ISTSAT-1 in the figure) goes over the first GS (GS1 in the figure) it obtains its first transmission opportunity. During this time windows the ISTSAT-1 will transmit as much as possible of the data blocks until the link drops and it is forced stop. This pieces are then sent from the GS1 to the Mission Control Center (MCC) server through the Internet.

An important thing to notice is that even though, in this example, multiple transmission will occur over different links, in a distributed manner, each individual transmission is performed in the same way as a one-to-one transmission from section 5.1.

This process repeats two more time in step **B** and **C** until all data blocks are sent and reassembled in their final destination, the MCC server.

Another interesting possible example is the transmission of data to earth from a cluster of cubesats. This is illustrated in figure 9, where the blue spacecraft, in order to transmit the data to earth, needs to forward the data through the remaining spacecrafts. The non-dotted lines represent good quality links, which are spacecraft-to-spacecraft and earth communications, and the dotted lines represent low quality disruptive links, which are spacecraft-GS communications.

This example can also be solved by the distributed transmission procedure, where the first spacecraft (marked with an **A** in the figure) will first transmit the data to other spacecrafts that can reach earth, who will in turn, according to their transmission opportunities, forward the data to earth. The spacecraft selection depends on the route algorithm used, as explained in section 7. If two or more GSs are at transmission range of the spacecraft, the selection of which will be used will also depend on the route algorithm. Transmission collisions are avoided since all packets are addressed to a single receiver.

6. Implementation

Any software project should enforce a clearly defined set of methodologies, tools and procedures to be used in its development. This very important for the success of any engineering project, but particularly in a project like this, that not only features a lot of members working together, from different engineering and professional backgrounds, but also where any error or mistake can be fatal, like it is typically the case in space related projects.

In this section the complete methodology employed

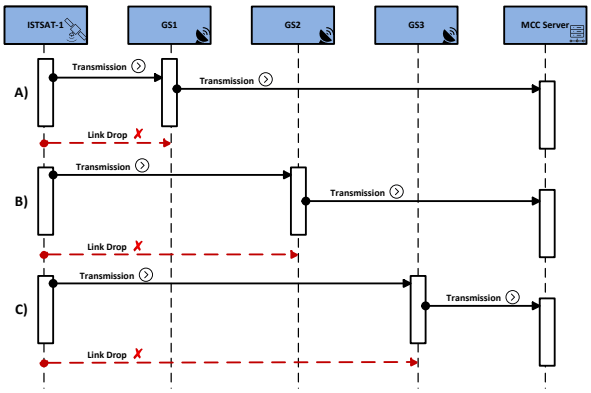

Figure 8: MCC Distributed Transmission

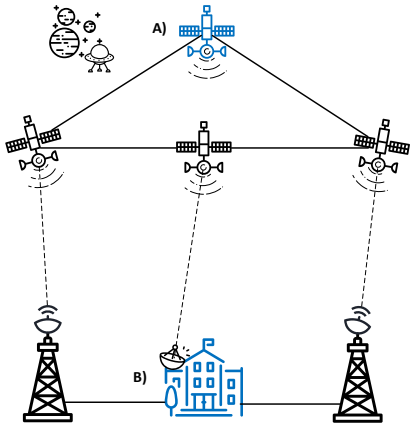

Figure 9: Distributed transmission - cubesat cluster example

in the development of this project will be explained in detail, some originating from the ISTSAT-1 project, others from the ESA FYS programme and some from the DDTP project itself.The implementation will also be explained in detail. This includes all the design decisions that were made to accommodate the specified requirements in the best way possible.

7. Software Architecture

The software development was divided in three modular sections: protocol, route and storage, as pictured in figure 10. These were implemented in the form of independent software libraries, containing the core protocol implementation, the route calculation, and the payload data storage module, respectively. This approach was used be the DDTP implementation is designed to be used in different kinds of systems, with architectures and capabilities.

The storage lib creates an abstraction level between the protocol lib and the selected storage medium, this results in an implementation that is not only designed for one storage method but can be more easily ported to multiple. In the illustrated example two mediums are available: RAM and flash storage.

The route lib offers an interface for accessing the routing table and dynamically updating its entries if such is desired, allowing, due to its modular nature, the implementation of multiple route calculation algorithms.

The protocol lib contains the actual software implementation of the DDTP protocol, including its algorithms and messages.

The application level software can be any software that needs to perform a DDTP transmission. The only requirement it needs to fulfill is to interface with the protocol library in order to perform a DDTP transmission.

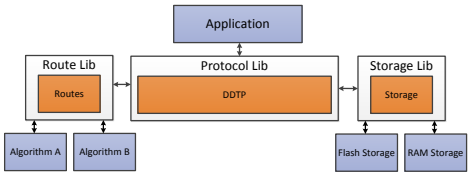

Figure 10: DDTP modules

7.1. Multi platform

One of the DDTP protocol requirements is the support of multiple Operating Systems (OSs). To achieve this, an architecture similar to the libcsp was used. All system related functionality is isolated in separated modules, one for each different OS. Every time a system related call is required by the software, instead of directly accessing a particular system interface (which would result in system lock-in), an internal DDTP interface module is called. This interface, called Operating System Abstration Layer (OSAL) in figure 11, will select the module of the particular system being called, make the necessary adjustments according the the system features and limitations, and execute the system depend functionality. This abstraction layer offer the possibility of easily porting the software to different platforms by only creating a new platform specific module that implements the specified interface. Without this technique it would be necessary to perform deep and unclear modifications to the source code, with a high probability of creating new problems and bugs. The compilation procedure was designed so that one is able to only include the modules corresponding to the necessary platforms, thus reducing the compiled executable size and memory footprint. Despite the OSAL interface, an effort was made to avoid using functionalities that were not available in all the targeted platforms, since doing so would require the development of this func-

tionalities in the modules where it was unavailable. Particularity the FreeRTOS platform is much slimmer in terms of functionality than the remaining platforms.
This design follows the common mentioned software development principle: The code should be designed to be extended, not modified.
Modules for three different platforms were developed, which will be explained bellow, but more could be easily created.

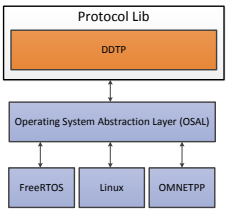

Figure 11: DDTP software platform architecture

7.2. Multi Network stack
The DDTP protocol is being developed under three different network stacks: TCP/IP, CSP/RDP and OMNET's simulated TCP/IP stack. This is illustrated in figure 12 where, in similarity to the approach used for the multi-platform problem described in section 7.1, an abstraction layer was used. This layer was named Network Protocol Abstration Layer (NPAL) and it abstracts every access between the DDTP protocol and its lower network stacks. This allows one to easily extend the project with new network stacks, add and remove stack modules without breaking the project (something very useful for memory constrained systems) and have multi stack compilations. This modular approach also allows one to test each module in isolation, something very useful for pinpointing problems early on.

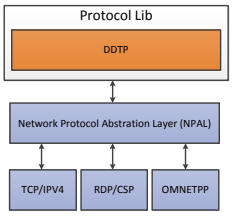

Figure 12: DDTP - Network Abstraction Layer

Multiple network stacks are necessary for this project since communications will travel over two very distinct mediums, requiring different protocols. In figure 13 the multiple stacks are illustrated, where depending on the different medium, either the RDP/CSP or TCP/IP stacks will be used.

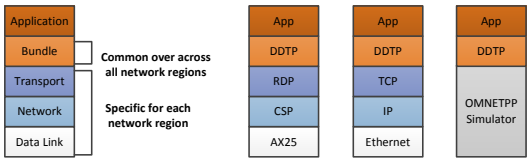

Figure 13: DDTP Network stack

The CSP/RDP stack is useful for space-link communications, like the ones between GSs and spacecrafts or spacecraft-spacecraft. This is due to its simplicity and low overhead. In this scenario the major concern is the quality of the link and not its congestion, in contrast with the typical earthly communication scenario like Internet connections.
The TCP/IP stack is, on the other hand, appropriate for Internet connections and is being used for the purpose of inter connecting GSs. Both TCP and RDP protocols offer guaranties of integrity, message order and delivery acknowledgment. These are the minimal requirements for a network stack running the DDTP protocol.
The OMNET stack is only used for testing, debuging and performance evaluation purposes. It comprises of an simulated TCP/IP stack that runs inside the OMNET simulator.
A single transmission can flow trough more than one stack as long there is a multi-stack node, like it is illustrated in figure 14. The nodes with multi stack capabilities are called gateways and are necessary for briding the two communication mediums.

8. Related work - Saratoga

Saratoga[15][14] is named after the USS Saratoga[5], and was first developed at Surrey Satellite Technology Ltd (SSTL) to download imagery from its remote-sensing Disaster Monitoring Constellation (DMC) satellites.
Saratoga is a very scalable, peer-to-peer reliable transport protocol, capable of transferring very large files (exa-exabytes - 2^{128} B) guaranteeing error-free data delivery using a Selective Negative

[5] `https://web.archive.org/web/20080328084822/http://www.chinfo.navy.mil/navpalib/ships/carriers/histories/cv03-saratoga/cv03-saratoga.html`

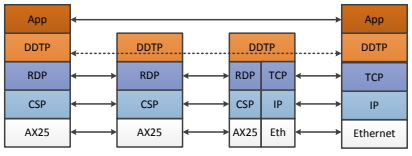

Figure 14: DDTP Multi stack transmission

Acknowledge (SNACK) logic.
Saratoga does not use any congestion control, it just sends the data at the highest possible rate in the link, with no regard for other flows. This is acceptable since space-links are typically dedicated.
It is a protocol based in User Datagram Protocol (UDP) (can also be used with UDP-lite to minimize checksum computation overhead), designed to send data packets as fast as possible (since it is designed for private, low congestion links there is no need for congestion control) over several hops on privately owned links. It is also capable of transmitting over links with very long propagation delays since no forced timeouts are used.
Saratoga uses a 32 B DATA message header, another 8 B for the UDP header, 20 B for the Internet Protocol (IP) header and another 18 B for the Ethernet header, for transmitting a payload of 1422 B (assuming a normal sized ethernet frame of 1500 B), offering a payload ratio of ∼95 %.
The Saratoga protocol was developed with design goals of being very fast; to use cheap, reliable and robust internet technologies; provide more speed than Transmission Control Protocol (TCP) can offer; not caring about congestion, because it is to be used as a single flow per link with no competition; support very big data file sizes; simultaneous delivery to multiple receivers. Two example implementations of the Saratoga specification are being developed, but are still incomplete.
The first one is a Perl implementation[6], that was written for interoperability testing, to validate the Saratoga specification. It was not designed with performance in mind, and like so, it is not suitable for a real scenario deployment, only for testing other implementations.
The second one is a C++ implementation[7], it is freely available for use and implements most of the Saratoga version 1 [2] (the latest one) specification. It is still incomplete but, unlike the perl implementation, it is intended to be suitable for real world scenarios.

9. Results

In order to validate and quantify the reliability, functionality and quality of the designed solution, a set of tests were performed. Since the characteristics necessary for testing a distributed DT protocol (such as high latency, high error rate, link disruptions, distributed network) are difficult to emulate in laboratory, the OMNET emulator was used to create a simulated network.
To add more value to the obtained values and compared them to the current available solutions, a publicly available saratoga reference implementation was evaluated in the same scenarios. An unmodified copy of its C++ implementation was used at revision *56b9f1*.
In figure 15 the simulated topology used for the following tests is illustrated. Each node of the network is connected to each other by a 10 KB/s channel, where links can be disconnected during a transmission, through scheduled events, or even before the simulation (forcing a transmission to follow a particular route). All nodes in the topology are running the same DDTP implementation, with same the upper layer software (see figure 11). They do however contain different route and network configurations, since they are placed in different network positions. It should be noted that node A and B are the only two nodes that contain more than one link, and so, have the possibility of using another network route.

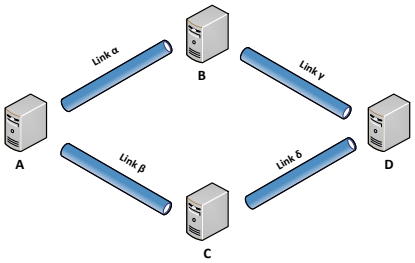

Figure 15: Testing simulation topology

10. One to one

In order to measure the performance of both protocols in a one-to-one scenario, using the topology illustrated in figure 15, multiple transmissions were performed between node A and B, exclusively over link α (link β was disconnect), with multiple data payload sizes. All the values obtained in this section were captured in the network interface of node B.
In figure 16 the DDTP and Saratoga transmissions are illustrated, where one can observe that the transmission time increases linearly with the payload size. In this scenario the Saratoga protocol is clearly faster, which is to be expected since it uses less metadata due to its decreased functionality compared with DDTP. However, even thought the difference is relatively small, this means that in a exclusively (combinations are possible) one-to-one scenario, a simpler protocol like Saratoga is preferable.
DDTP achieved an average throughput of 8.909 KB/s and Saratoga of 9.295 KB/s.

10.1. One to one - Delay tolerant

Another feature that requires verification is the DDTP DT capabilities. Again, the transmissions

[6] `http://saratoga.cvs.sourceforge.net/viewvc/saratoga/saratoga-v1-perl/`
[7] `http://sourceforge.net/p/saratoga/saratoga-vallona/ci/default/tree/`

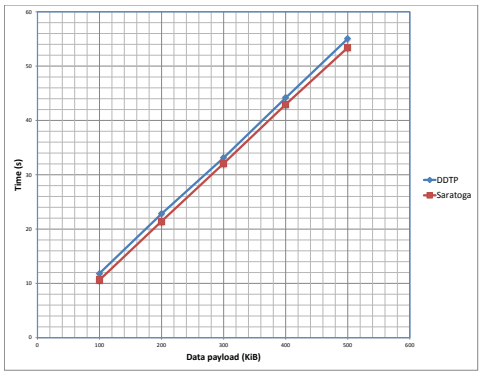

Figure 16: DDTP vs Saratoga - One to one

were performed between node A and B (15), exclusively through link α (link β was disconnect), with a 200 KB data payload, and all traffic was captured at the network interface of node B.
In figure 17 the DDTP transmission throughput is displayed in function of the time, where each point in the chart corresponds to a 1 s average of the throughput. A link disruption was scheduled at the 5 s mark, causing an unexpected interruption in the transmission. This forces the transmission to pause, waiting for the link re-connection, since no other routes are available. As can be observed in the mentioned figure, from the 5 s mark to 53 s no data was transmitted. At that point the link reconnection is detected and the transmission is resumed, finishing at 72 s.
As was desired, the property of being able to pause and resume a transmission according to the link availability was validated. This is one of the core requirements of the developed protocol.

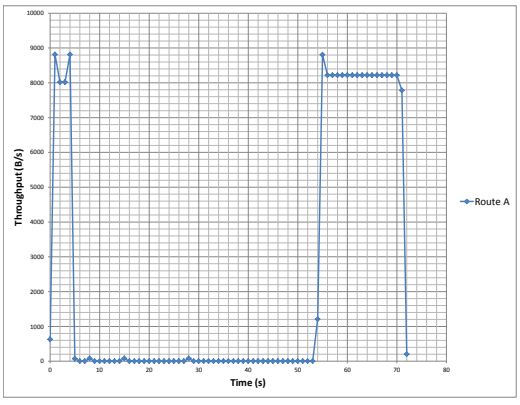

Figure 17: DDTP DT transmission

In figure 18 the same experience, in the same conditions, was repeated with the Saratoga implementation. Here the link is once again disconnected at the 5 s mark, reconnecting at 20 s. Similarly to DDTP the transmission was able to pause and resume accordingly to the availability of the link.

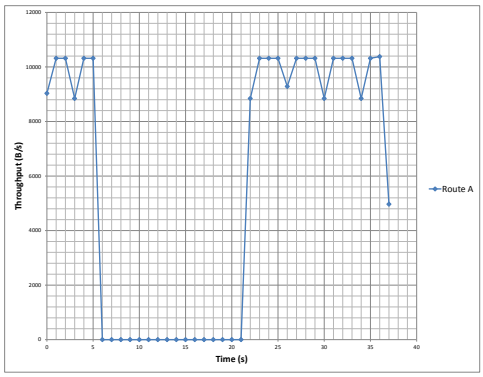

Figure 18: Saratoga DT transmission

11. Distributed

To validate DDTP main feature, its distributed property, a transmission of 200 KiB was performed between node A and D, using the topology illustrated in figure 15. All links and nodes were connected at the beginning of the simulation, with a link disruption scheduled for link α. Node A has two routes for reaching node D: Route A through link α; and route B through link β. Unlike what happened in section 10.1, where only one route was available and so node A was forced to pause the transmission, when the disruption occurs the transmission should resume through route B.
In figure 19 this experiment is illustrated, where each point in the chart represents a 1 s throughput average. When the link α is disconnected, at the 9 s mark, the transmission through route A stops. At this point any current solution would proceed by simply pausing the transmission and waiting for the link restablishment. This is the behavior observed in figure 18 by Saratoga. Unlike Saratoga, however, the DDTP resumes the transmission through route B as soon as it detects that route A is no longer available. The result of this is the full transmission finishing in under 30 s. The simpler DT approach used in section 10.1, with the same data payload, resulted in a 72 s transmission time, this corresponds to a huge improvement on DDTP part. It should also be taken into account that for bigger link disruption times this difference your higher, increasing linearly with the link disruption time.

12. Conclusions

A cubesat is a very resource constrained type of spacecraft, with low energy reserves, small solar panels, and low communication and processing capabilities. Due to this, the development of a cubesat offen requires very optimized solutions. The ISTSAT-1 mission is to perform a feasibility study of the use of cubesats for receiving ADS-B sig-

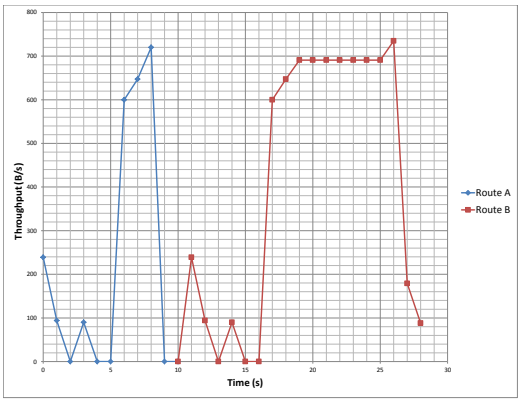

Figure 19: DDTP distributed transmission

nals from traveling airplanes. A safety system that broadcast information from each airplane. It then needs to transfer the processed ADS-B information to earth, through one or more GSs. Since cubesats are typically launched in LEO, they have a very fast traveling speed (close to 8 km/s), that results in a very small time window (under 20 min) to communicate with the GSs on earth. Current solutions solve this problem by operating in a DT fashion.

This project proposes a new approach, the development of a distributed DT protocol, capable of performing the same tasks already being solved by the current solutions but in a much more optimized way, increasing the overall speed at which data can be received. This solution is not only appropriate for spacecrafts like the ISTSAT-1, that simply wish to transmit mission data to a server, but can also be used for more complex networks like spacecraft clusters with Inter Satellite Links (ISL) and the simultaneous transmission of data to multiple receivers on earth, like a Wireless Sensor Network (WSN).

Four application scenarios were defined, and these encompass the major requirements necessary. From these four, only two are currently being solved by the existing solutions: one-to-one and multi-to-one transmissions. The developed solution was designed to not only solve the first two but also the new ones: one-to-multi and distributed transmissions.

The experimental results showed that using DDTP will result in much faster transmissions in two of the scenarios.

When multiple links are available (distributed scenario) DDTP is able to make use of these links and avoid completely pausing the transmission when a disruption occurs. This offers a very significant performance improvement over current solutions which simply operate in a DT fashion in these scenarios. This is the use case of the ISTSAT-1 project.

When multiple transmission destinations are available over a single link, DDTP also manages to reduce very significantly the transmission times. While current solutions simply perform several one-to-one transmissions to solve this problem, DDTP multicast capabilities allow the transmission to reach all destinations simultaneously. This is the use case of a WSN whose nodes are receiving a firmware update over a space link.

The first two scenarios, one-to-one and one-to-multi are solved by both DDTP and the current solutions, but due to the increased metadata usage in DDTP current solutions are actually slightly faster, in the one-to-one test Saratoga was able to finish the transmission 1.8 s faster than DDTP. However, purely one-to-one or one-to-multi scenarios are not the use case for which DDTP is designed for.

Future work should include the testing of DDTP in real world scenarios, including its integration with the remaining systems of the ISTSAT-1 cubesat. More complex routing algorithms could also be developed, potentially increasing transmission speeds even further. One possibility would be having DDTP collect information from the module responsible for determining the ISTSAT-1 location (the Altitude Determination and Control System module). This would allow DDTP routing algorithm to predict when links would become available/unavailable and make choices based on this information. DDTP power usage onboard of ISTSAT-1 should also be analyzed and, if necessary, optimized.

References

[1] R. F. Afonso. Sistema de Gestão e Determinação de Atitude do Engenharia Electrónica Júri. page 74, 2016.

[2] C. S. W. E. C. Jackson, W. Ivancic and L. Wood. Saratoga: A Scalable Data Transfer Protocol. RFC Draft, RFC Editor, March 2012.

[3] CalPoly. Cubesat design specification. *The CubeSat Program, California Polytechnic State ...*, 2009.

[4] O. N. Challa and J. McNair. CubeSat Torrent: Torrent like distributed communications for CubeSat satellite clusters. *Proceedings - IEEE Military Communications Conference MILCOM*, 2012.

[5] P. Coelho. ISTNanosat-1 Quality Assurance , Risk Management and Assembly , Integration and Verification Planning. (May):107, 2016.

[6] J. Ferreira. ISTNanosat-1 Heart. Master's thesis, IST - Instituto Superior Técnico, 2012.

[7] E. N. W. G. G. Camarillo. Peer-to-Peer (P2P) Architecture RFC. RFC, RFC Editor, 2009.

[8] P. M. Gameiro. Distributed Delay Tolerant Protocol. page 6. 15th International Symposium on Wireless Communication Systems (ISWCS), 2018.

[9] A. Mccook. On the right path. *Nature*, 2011.

[10] A. Schlesinger, B. M. Willman, L. Pitts, S. R. Davidson, and W. A. Pohlchuck. Delay/Disruption Tolerant Networking for the International Space Station (ISS). *IEEE Aerospace Conference Proceedings*, (October 2010), 2017.

[11] Secure World Foundation. *New Actors in Space*. 2011.

[12] Y. Song, L. Liu, and J. Chen. A Temporal Graph Model based Power Aware Routing Algorithm in Deep-space Networks. 2017.

[13] A. H. L. T. R. D. K. S. K. F. V. Cerf, S. Burleigh and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838, RFC Editor, April 2007.

[14] L. Wood, W. M. Eddy, W. Ivanèic, J. McKim, and C. Jackson. Saratoga: A Delay-Tolerant Networking convergence layer with efficient link utilization. In *2007 International Workshop on Satellite and Space Communication, IWSSC'07*, 2007.

[15] L. Wood, C. Smith, W. M. Eddy, W. Ivancic, and C. Jackson. Taking Saratoga from space-based ground sensors to ground-based space sensors, 2011.