Journal of Applied Mathematics & Bioinformatics, vol.3, no.1, 2013, 163-179
ISSN: 1792-6602 (print), 1792-6939 (online)
Scienpress Ltd, 2013

An Efficient Impelementation of the Bundle Security Protocol for DTN-enabled Embedded Devices

Christos Tselikis[1], Athanasios Poulakidas[2], Aggelis Aggelis[3] and E.G. Ladis[4]

Abstract

As the implementations of the Bundle Protocol (RFC 5050) reach remarkable maturity, an increased interest is now appearing to fully implement the DTN security features as specified in the Bundle Security Protcol (RFC 6257). In this regard, in this paper we present our BSP implementation suitable for embedded devices with scarce resources (limited prcocessing, memory, energy and transmission capabilities). Our BSP implementation enhances the existing security features in DTN2(v2.8). We analyze the functionality and the performance of our secure DTN2 stack via experimental means. Namely, we conducted a series of test cases in our heterogeneous TCP/IP-based test-bed. The

[1] Hellenic Aerospace Industry S.A., e-mail: tselikis.christos@haicorp.com
[2] Hellenic Aerospace Industry S.A., e-mail: poulakidas.athanasios@haicorp.com
[3] Hellenic Aerospace Industry S.A., e-mail: aggelis.aggelis@haicorp.com
[4] Hellenic Aerospace Industry S.A., e-mail: egladis@haicorp.com

Article Info: *Received* : November 1, 2012. *Revised* : January 30, 2013
Published online : March 30, 2013

latter allowed for experimentation regarding bundle transmission via different network interfaces (high bit-rate wire-line and low bit-rate wireless links) and with different types of interconnected DTN-enabled devices. The results prove that the security developments impose tolerable communication overhead and reveal the DTN performance factors.

Mathematics Subject Classification: 68M12
Keywords: Bundle Security Protocol; embedded devices; test-bed; performance

1 Introduction

The Bundle Protocol (BP) defined in DTNRG RFC 5050 supports interoperability across different networks, creating an overlay network where two or more endpoints can seamlessly exchange information overcoming different transport and network layers. The Bundle Security Protocol (BSP) is defined in IETF RFC 6257 [1] and specifies security features for the Bundle Protocol in order to provide DTN security services. The BSP describes four mandatory cipher-suites but does not impose any restriction on the cipher-suites that can be used; however there is a clear need for efficient security implementations that can be ported to embedded environments successfully meeting the fundamental requirement to authorize the access to the DTN infrastructure. In this paper we present an enhancement of the DTN2 Reference Implementation v2.8 with implantation of the security features specified in BSP: end-to-end bundle encryption with Payload Confidentiality Block (PCB), end-to-end bundle signature with Payload Integrity Block (PIB), hop-by-hop Bundle Authentication Block (BAB) signatures and RSA public key encryption, excluding the Extension Security Block (ESB). The functionality and the performance of the secure DTN stack was validated with a series of test cases conducted in our heterogeneous test-bed.

2 Related Work

1) DTN2 [2], the widely used Reference Implementation (RI) in C++, tested on Linux (x64 and 64-bit x86) and Mac OS X (PPC and x386). DTN2 supports TCP, UDP, NORM, AX.25, LTP and Bluetooth convergence layers. Also, DTN2 supports table-based routing, Bonjour, ProPHET, DTLSR, epidemic routing and elementary secure DTN features.

2) ION [3], the second RI is an implementation of BP, LTP, CFDP, and AMS in C, tested on Linux, OS X, FreeBSD, Solaris, RTEMS, and VxWorks. ION is interoperable with DTN2 and supports TCP, UDP, and LTP convergence layers. ION is the only software to implement the Contact Graph Routing (CGR) algorithm, which is considered the most suitable for space contacts. ION is suitable for embedded environments due to reduced footprint. With regard to security, from open-source version 3.0.0 onwards, implementations for the BAB, PCB and PIB security blocks are included in ION, tested over UDP/IP.

3) IBR-DTN [4], a very portable, slim and extensible implementation in C++, tested on Linux (x386 and MIPS) and runs on every system with OpenWRT. It claims full support for BSP and includes TCP, UDP, and HTTP convergence layers along with experimental support for various Internet Drafts. IBR-DTN was also ported to run on an iMote2 sensor running a modified version of Linux. This port supported only IEEE 802.15.4 (LowPAN) as a convergence layer. IBR-DTN supports table-based routing, UDP and TCP discovery, IP neighbor and epidemic routing with an efficient bloom filter.

4) ByteWalla [5], an implementation in Java (over TCP) for Android devices; it was later ported in pure Java for Microsoft Windows and Linux systems. Supports static and dynamic routing (Prophet). With regard to security, implements the BSP PCB security block with AES in Galois Counter Mode (GCM).

5) CONDROID [6], a design of a DTN/WSN Gateway and Java applications running on Android devices.

6) POSTELLATION [7], a DTN implementation, running on Windows, MacOSX, Linux and RTEMS. Included applications are dtnping/dtnpong, dtnsend/dtnrecv, HTTP/HTTPS Proxy, RSS news service delivery such as NASA news over DTN.

3 Requirements

The identified security requirements span three basic levels, namely the bundle, the storage and the system level as described in more detail in the sequence.

3.1 Security Requirements at the Bundle Level (SBR)

SBR-01: Data confidentiality. Except from encrypting the bundle payload with the PCB security block it is also recommended to transmit the Bundle Encryption Key (BEK) encrypted inside the PCB block. The encryption of the BEK can be symmetric or asymmetric.

SBR-02: Data integrity. The modification of messages is an attack against data integrity and the protection is achieved with message hashing (MD-5, SHA1, and SHA-2), integrity check values and digital signatures.

SBR-03: Data origin and data integrity: the hop-by-hop BAB security block ensures the authenticity of the bundle origin and the integrity of the bundle payload.

SBR-04: Authorization. Access Control Lists can be used as a mechanism to grand permissions to applications and users to access the DTN infrastructure. Address screening is another low-cost mechanism which prevents the unauthorized usage of DTN resources and which mitigates attacks such as flooding and even Denial of Service attacks.

SBR-05: Accountability. In the Bundle Protocol definition the accountability and traceability are actually provided with the bundle status reports. However, in security terms accountability demands for permanently stored audit logs.

SBR-06: Non-repudiation. It is achieved when time-stamped digital signatures (and PKI certificates) are used. When signing with the BAB, the header which contains the timing information (bundle creation time and bundle expiration) is also signed. Hence, this timing information inside the BAB signature can be used for non-repudiation purposes.

SBR-07: Data confidentiality and DTN entity authentication. Confidentiality by itself can not countermeasure complicated attacks, such as a Denial of Service attack and therefore only when assisted with authentication mechanisms can it be proved more efficient. One algorithm that satisfies this security requirement is the AES in Galois Counter mode (GCM). AES-GCM takes as input the plain-text and the Initialization Vector (IV) and gives as output the cipher-text with an ICV tag that proves the integrity of data. Moreover, the receiver authenticity can be proved by encrypting the bundle encryption key with the public-key of the receiver.

3.2 Security Requirements at the Storage Level (SSR)

SSR-01: The node must provide a persistent dedicated space (e.g., a database) that is only accessible by the DTN software (BP agent) for storing its bundles and optionally for storing some administrative data. No other service or process may be able to access this dedicated space. From a security standpoint, the disk space will be divided in a few partitions. The partitions will be dedicated for bundle storage either directly in the file system or through a database system.

SSR-02: The node must provide a secure storage area for long-term storage of keys and critical configuration parameters. Presumably, this storage area will be in the order of some KB.

SSR-03: Policy Enforcement Point (PEP) is the entity that checks the security destination of the received bundles and interrogates the Security Policy Database (SPD) to obtain the corresponding key and other security-related parameters (e.g., the cipher-suite used) for a given DTN source-destination pair.

SSR-04: The node should provide a security policy database for storing entries with the policies/security relationships per each DTN source/destination pair (in DTN version 2.8 the security policies are defined per bundle which is

quite elementary).

SSR-05: The node must be reliable in terms of (single) disk failures.

3.3 Security Requirements at the System Level (SSYSR)

SSYSR-01: The node must contain as much DTN software as to support its mission. No additional or unused software modules will be available in the node as to avoid possible exploitation through unused or non-configured modules and software libraries.

SSYSR-02: The node must run the DTN software as a separate user account, preferably a non-privileged one, in order to minimize the surface for elevation attacks and exploits of the system through the DTN.

SSYSR-03: The node must collect detailed logs of its DTN operation, for both debug and auditing purposes. The logs may be rotated and may be configured to be send as bundles to a central logging facility for long-term storage.

SSYSR-04: All system accesses (local or remote) are logged to the syslog facility. The access logs may be transmitted to a configurable central facility for long-term storage and may be rotated, for example, every 3 months.

4 Design and Implementation

DTN2 version 2.8 software distribution for a Linux environment covers a large portion of our identified requirements. Thus, it is a wise option for realizing the secure DTN node. However, there is some functionality that is not offered or supported by the DTN2 RI. This functionality, as summarized below, was contemplated necessary and was implemented in the secure DTN2 stack:

1) **Minimization of the DTN2 size**. The default installation of DTN2 v2.8 occupies 23MBytes. We reduced this significantly achieving 2.6MB with the baseline development and we achieved 3.1MB when including the BSP implementations and the OpenSSL libraries. This is roughly an eight-fold decrease of the DTN2 executable size. Adding the LTPlib,

4MB are reached. External dependency remains only for the tcl and the Berkeley DB library.

2) **Implementation of missing BSP functionality in DTN2 RI**, including:

a) **Support for pe-link BAB keys**. Each link of the DTN infrastructure is authorized with different BAB keys.

b) **Support for PIB functionality with RSA digital signatures**.

c) **Support for enhanced PCB functionality**. Encryption of the payload with the Bundle Encryption Key (BEK) plus symmetric encryption of the BEK plus protection of the encrypted BEK inside an RSA envelope (with public key encryption).

d) **Integration of OpenSSL** (1.0.1c, May 2012) [8].

3) **Implementation of a per-destination bundle storage quota mechanism** as a countermeasure to bundle flooding attacks. According to this mechanism, the DTN2 command: storage set payload-quota <bytes> is enhanced to the command: storage hostquota <host> <bytes> so that If there is enough storage space for a specific destination host the bundle is accepted, otherwise it is not and a BundleProtocol REASON-DEPLETED-STORAGE reject reason is provided to the sender.

4) **Separate storage of the DTN daemon logs** in a different file system than that used for storing the bundles and periodic rotation of the log files for improved performance (the corresponding definitons are added in the DTN configuration file).

5) **Use of RAID-1 configuration** for bundle payload data redundancy.

5 Verification Test-Bed

Our TCP/UDP/IP test-bed is a collection of heterogeneous devices with different network connections, namely FE, GE, 802.11/b/g/a/n and ADSL (offers Internet access).

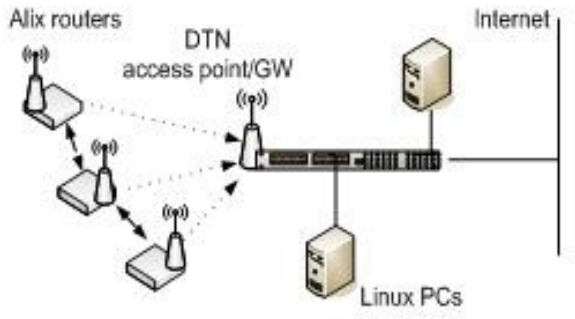

Figure 1: The TCP/IP based verification test-bed

The most essential part to describe in detail here is the different types of the devices that we used to install, configure and verify the secure DTN functionality:

*Type A*1. Linux-based PC machines (Ubuntu 10.04) each one with motherboard Intel D945GCLF2, Atom 330 dual core, 1.2 GHz, 1024 MB RAM, 80 GB HDD SATA II, Sound Card on Board 7.1 HD Audio, 4, 2 USB, Parallel, Serial, LAN 10/100 FE cards.

*Type A*2. Robust Linux-based PC machines (Ubuntu 10.10) with GE network card, Intel Pentium D 3.4GHz, 80GB storage and 1GB DDR2 physical memory.

Type B. VM with virtualization software the VMware Player 4.0.1. In this case the hosts are Windows 7 Professional 64-bit with SP1 machines and with Intel Core i7 950 3.06 GHz with 1066 MHz system bus, 12 GB DDR3-1600 (3x4 GB), OCZ Vertex2 120 GB SSD (system), one WD RE4 1 TB SATA 3 Gbps and Realtek Gigabit Ethernet onboard adapter (Gigabyte GA-X58A-U3R). This type of host is used for the measurements of the maximum achievable TCP/IP throughput via virtual connections and without DTN.

Type C. Linux-based embedded devices, namely Alix 2D2 boards with CPU: 500 MHz AMD Geode LX800, DRAM: 256 MB DDR DRAM, 4 GByte Storage CompactFlash, 44 pin IDE, DC 7V-20V, 2 miniPCI, 2 Ethernet channels, serial port, dual USB port, board size 6x6, firmware tinyBIOS. We installed the Voyage Linux embedded distribution is on this type of device.

Type D. Ground router with embedded characteristics, namely Arinfotek

Teak 5020L-280 device, Industrial-Grade 1U Chassis 8 Gigabit LAN, 2 Pairs of LAN By-Pass CPU in Intel Long-Term Support Roadmap DDR-3 Memory for High System Performance Intel LAN Chips for Best Reliability and Throughput Flexible Expansion PCI-32 slot Up to 2x 2'5" HDD for Large Storage need Optional 2 X 20 LCM for HMI. We used a read-only CF disk to load the operating system and the DTN stack and one faster and of larger capacity SSD disk (read-write) to permanently store the bundles. The Voyage Linux embedded distribution was also installed on this type of device. Further, we proceed on the equipment of Teak with two small-size SSD disks with RAID-1 configuration in software for data redundancy.

5.1 Security Tests

We tested the security developments with most of the standard DTN2 applications, namely dtnperf2, dtnsend-dtnrecv, dtncp and dtnchat so that it is proved that the developments are independent of the end-user application. In addition, we configured and partitioned the TCP/IP test-bed shown in Figure 1 in many different ways in order to accomplish the various validation tasks. In the configuration shown in Figure 2 we used two Linux-based machines (*Type A*1) which were inter-connected via the DTN router-relay device. All three nodes in Figure 2 were loaded with the DTN2 stack. We supported static routing at the DTN level over TCP ON DEMAND links.

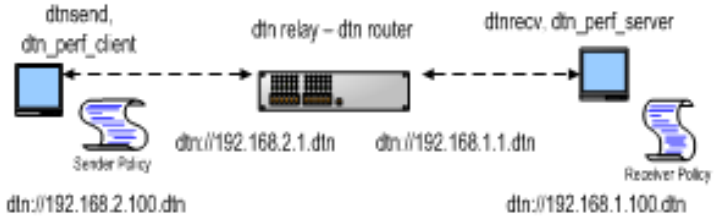

Figure 2: The test-bed configuration for the security tests

With the configuration of Figure 2 we conducted a series of validation tests

that prove that our security implementations are functionally correct. We present here the most important test cases covered.

1. Multi-hop security with BAB. The two end-nodes shown in Figure 2 have the implemented security features installed and the DTN daemons are up and running. The same holds for the intermediate router node. All three nodes have activated the BAB policy. The intermediate node verifies the received SHA1-based HMAC BAB block (the necessary 20 bytes long HMAC key is pre-loaded or searched in the key database per source node) and if it is valid substitutes it with a newly created BAB signature (hashed with a different HMAC key, according to our support of per-link BAB keys) and forwards the bundle, otherwise it discards it. In contrast, the DTN2 reference implementation (at least up to version 2.8) the same symmetric BAB key has to be used by all nodes in the path from the source to the final destination. This particular test case verified the correct implemenation of the BAB-HMAC1 mandatory BSP cipher-suite with the per link BAB key implementation.

2. End-to-end security with PCB. The two remote nodes in Figure 2 (source and destination) have the implemented security features installed with the PCB security policy activated and the DTN daemons are up and running. However, the intermediate DTN node has no security policy defined at the DTN level (i.e., is *security unaware* node which forwards without processing the secured bundle). According to the implemented PCB policy, the source secures the bundle payload by applying the AES in GCM mode with 128-bit key and adds the PCB block after the bundle primary header. A 16 bytes long randomly generated symmetric key is used as for the Bundle Encryption Key. The AES in GCM mode gives as output the payload ciphertext and the ICV tag. In contrast to DTN2, our PCB implementation uses a second symmetric key (randomly created), the Key Encryption Key (KEK), which is used for the encryption of the BEK using the AES-128 in CBC mode. The encrypted BEK is further protected with RSA public key encryption and the resulting RSA envelope is included in the PCB block. In total, for a 4096-bit RSA key the PCB length is 564 bytes and for a 1024-bit RSA key the PCB length is 180 bytes. On the contrary, 16 bytes are needed for the AES-128 PCB in DTN2 RI and the 372 bytes in the Bytewalla AES-GCM PCB implemenation [5]. The encrypted bundle along with the PCB is sent to the final destination address via routes that we configured statically at the DTN level. At the

destination end, first the authenticity of the receiver has to be proved (this is achieved if the final destination holds the correct RSA key). Then the final destination verifies the integrity of the received bundle by checking the validity of the ICV tag. In sequence, the BEK sent by the remote source is recovered and with the BEK the destination node decrypts the payload ciphertext and acquires the originally transmitted message. The destination private RSA key, namely the *decrypt.pem* certificate in .PEM format, is pre-installed at the local path */home/user/dtn/var/certs*. This particular test case verified the correct implemenation of the PCB-RSA-AES128-PAYLOAD-PIB-PCB mandatory BSP cipher-suite.

3. End-to-end security with PIB. Two remote nodes connected via the network (in more than one-hop distance) have installed the secure DTN stack with the PIB security policy on. The DTN daemons are up and running. The same holds for all the intermediate DTN nodes however the latter have no security policy activated (*security unaware nodes* which forward without processing the secured bundle). The source node secures the bundle payload by applying the RSA digital signature operation with its own RSA private key. In our implementation when the size of the RSA digital signature is 128 bytes, the PIB occupies a block of 132 bytes, namely 128 bytes for the signature plus four bytes for describing its size. On the contrary, 32 bytes are used for the SHA2 digest in DTN2 RI (no signing with RSA key is performed). The private RSA key of the source (namely *sign.pem*) is stored at the local path */home/user/dtn/var/certs/*. The final destination uses the public RSA key of the source to verify the received signature, namely $dtn - hostname - verify.pem$, which is stored at the local path */home/user/dtn/var/certs/*. All the tests in this particular test case verified the correct implementation of the mandatory PIB-RSA-SHA256 BSP cipher-suite.

5.2 Performance Tests

In this section we present the results of DTN performance tests conducted with the dtn-perf(v2) tool and the *Type D* device (Teak ground router). Our aim was to evaluate the BSP security overhead on DTN performance (goodput and delay). Since the security blocks (e.g., PCB) and the bundle header are constant in their size, the security overhead decreases with the bundle payload

size p according to $1/p$ [5]. However, we are interested in scenarios where the bundle are injected into the network and we want to compare the DTN goodput that can be achieved by DTN-enabled nodes when processing bundles of variable sizes *with* and *without* the BSP policy implementations. To this end, we cross connected the Teak router device (dtn-perf server) via a GE connection with a bundle generator machine. At this point we present experimental results regarding the AES speed in CBC mode when changing the key and file sizes. For the AES measurements we used one standalone *Type A*1 device and OpenSSL 1.0.1c.

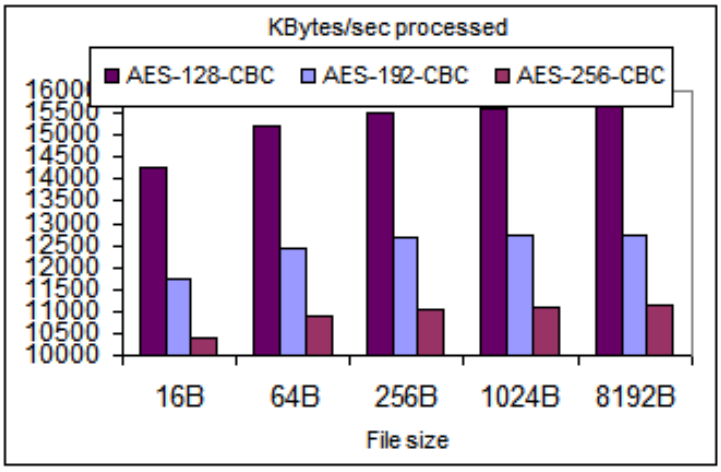

Figure 3: The AES-CBC speed with variable key sizes

The results shown in Figure 3 justify our choice of AES-128 in CBC mode in our PCB implementation.

In the sequence, we describe results for the two basic scenarios that we examined regarding the DTN goodput performance:

1. DTN goodput with no link disruptions. In this case the dtn-perf client was a *Type A*2 robust PC cross-connected with the ground router (dtn-perf server) via two GE network cards. We sent bundles and received per bundle delivery reports with and without the security policies and we compared the reported goodputs. Specifically, the client machine generated two types of bundle flows:

a) A workload with a total data volume equal to 1 Gigabyte which was segmented according to the bundle payload size chosen (100MB, 200MB and 300MB). This feature is offered by dtn-perf(v2). The resulting continuous stream of bundles was sent to the server (no setting for congestion control and no custody transfer request was used at the bundle generator side) and one per bundle delivery status report was returned back to the client. The permanent bundle storage in these experiments was the file system at the server machine. The experiments with 1GByte workload were conducted with security (PIB + 1GB Workload) and without security (Plain DTN2 + 1GB Workload). The corresponding goodput figures were reported and logged at the client machine (in kbps) and these are illustrated in Figure 4 (the RSA key was 1024 bits long).

b) Single bundles with bundle payload size 100MB, 200MB and 300MB. Each single bundle was sent from the client to the server and a delivered status report was returned. The experiments were conducted with the security policies on (DTN2+BAB, DTN2+PIB and DTN2+PCB) and off (plain DTN2).

The goodputs achieved for those two types of interactions are shown in Figure 4 in which each point is the average of five measurements.

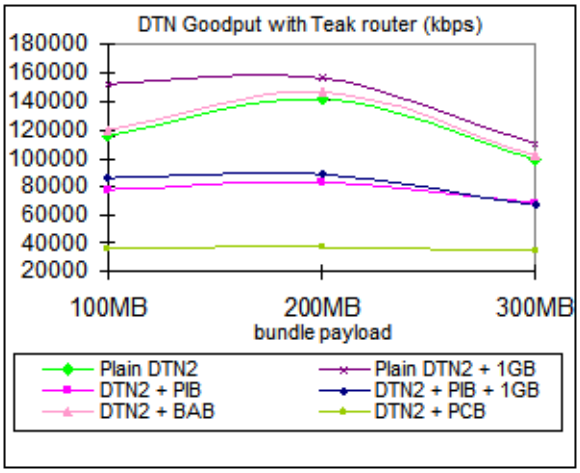

Figure 4: DTN goodput with and without security policies

In Figure 4 we observe that in the case of transferring a single bundle with

200MB payload the goodput achieved by the Teak device was maximized. In particular, for the plain DTN2 case (no security policy activated) that was 142 Mbps approximately. However, when the client sent a workload of 1GB, segmented in five unsecured bundles of 200MB each, the goodput was increased to 155 Mbps, thus achieving even better utilization of the network resources. The same was true when sending the 1GB workload with the PIB policy on. For bundle payload sizes larger than 200MB the DTN goodput (with and without security) started to degrade (compression can be used in order to decrease the security overhead). When the BAB policy was activated, the plain DTN2 performance was sustained - rather it was slightly increased for all payload sizes, as it is evident in Figure 4.

Table 1: The % DTN2 goodput degradation due to BSP policies implementation

Bundle size	100MB	200MB	300MB
BAB policy	+3.82	+4.03	+3.44
PIB poliy	-32.74	-41.43	-31.15
PCB policy	-68.56	-73.02	-64.46

Table 1 shows the goodput overhead imposed on DTN2 per security policy and per bundle payload size.

2. DTN goodput with link disruptions. In this case we introduced channel disruptions with a ratio of the link up-time/down-time to be no more than 4/3 and no less than 2/9. In more detail, we disrupted the link between the bundle generator and the Teak device with a random fashion, i.e., the link up time was randomly chosen between 1 and 2 minutes, while the link down time was randomly chosen between 1.5 and 4.5 minutes. As expected, depending on the random result, the bundle transfer delay can be normal (for example when a single bundle passes through at the first communication opportunity) or can be arbitrarily long (exactly due to the randomness of the time intervals). For example, in one test it took 560.7 seconds for a single 200MB bundle protected with the PCB block to be delivered with acknowledge and the respective goodput dropped to 2.853Mbits/second. In another test, a 100MB bundle was delivered normally within 22.3 seconds and the re-

spective goodput was 35.9Mbps. Another way to introduce link disruptions (propagation delay, BER, packet corruption, packet loss and link assymetry) is by using the NETEM Linux kernel tool. The Voyage OS which was installed on the Teak device has the NETEM readily available and DTN performance tests are being conducted under low, medium and high BER conditions.

In the sequence, we describe results for the bundle transfer end-to-end delay:

Table 2: Average DTN delay (milliseconds) in a point-to-point connection

Bundle size	10KB	100KB	2MB
DTN Net delay with 2MB workload	10071.91	735.82	231.477
DTN Net delay with one bundle	65.67	61.77	280.8
DTN API delay (loopback test)	37.12	17.34	80.09

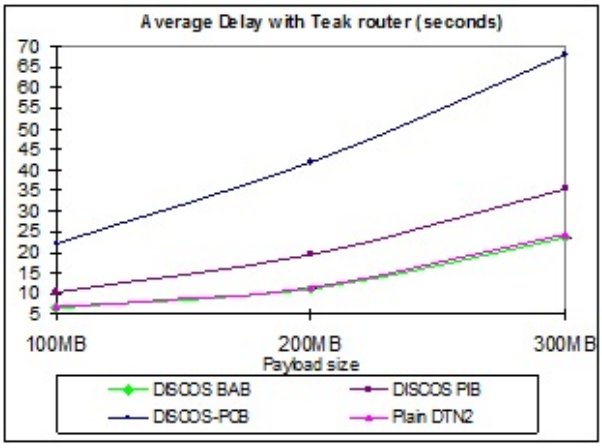

Figure 5: Average delay: plain DTN2 vs. DTN2 with BSP security policies

Table 2 shows the average delay experienced when transferring bundles with variable size between two *Type A* machines running plain DTN2. The two machines were connected via FE interfaces. Each delay figure in Table II is the average of ten measurements. It is worth noting that the one-bundle delay is minimized when the bundle payload equals the TCP socket buffer size (set to 100KB in our tests). The DTN API delay corresponds to the delay to

send and receive the bundles through the DTN stack measured in the loopback test case.

Figure 5 shows the average delay experienced when transferring bundles of sizes 100MB, 200MB and 300MB from the client to the Teak device with acknowledgement returned. The RSA key was 1024 bits long. We observe in Figure 5 that the security overhead with respect to the point-to-point bundle transfer delay was negligible when the BAB security policy was applied to the bundles. The PIB policy incured tolerable delay increase. The delay overhead was more evident when the PCB policy was activated.

6 Conclusion

A small-footprint version of DTN2 was developed with OpenSSL-based security enhancements suitable for embedded devices. Link authorization with per link BAB keys, protection of the Bundle Encryption Key with RSA public key encryption and protection from bundle flooding attacks with modification of the DTN2 storage quota mechanism were introduced to DTN2. The validation tests showed that when the asymmetric cryptosystem is used judiciously (i.e., small RSA key size and usage of RSA encryption during the authentication phase only) the DTN2 network perfrormance exhibits tolerable degradation.

Acknowledgements. The work presented in this paper was managed, supported and funded by the European Space Agency (ESA) project DISCOS (Distributed Information Storage and Communication in Outer Space) with members the Hellenic Aerospace Industry, the Democritus University of Thrace and the Industrial Systems Institute.

References

[1] S. Symington, S. Farrell, H. Weiss and P. Lovell, Bundle Security Protocol Specification, RFC6257, *Internet Society*, (May, 2011).

[2] M. Demmer, *DTN Reference Implementation*, Version 2, DTN Research Group, http://www.dtnrg.org/wiki/Code.

[3] http://sourceforge.net/projects/ion-dtn/.

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

[5] http://sourceforge.net/projects/bytewalla3.

[6] http://csd.xen.ssvl.kth.se/csdlive/content/condroid-project-overview.

[7] http://postellation.viagenie.ca/.

[8] http://www.openssl.org/source/.