Wolf-Bastian Pöttner, Johannes Morgenroth,
Sebastian Schildt, Lars Wolf

An Empirical Performance Comparison of DTN Bundle Protocol Implementations

Informatikbericht 2011-08

Institute of Operating Systems and Computer Networks
Carl-Friedrich-Gauss-Fakultät
Technische Universität Braunschweig

Braunschweig, Germany

Changelog

Version	*Date*	*Comment*
1.0	2011-09-16	Initial Version

An Empirical Performance Comparison of DTN Bundle Protocol Implementations

Wolf-Bastian Pöttner, Johannes Morgenroth,
Sebastian Schildt, Lars Wolf

IBR, Technische Universität Braunschweig
Mühlenpfordstraße 23, Braunschweig, Germany
[poettner|morgenroth|schildt|wolf]@ibr.cs.tu-bs.de

September 19, 2011

Abstract

In recent years, Delay Tolerant Networking (DTN) has received a lot of interest from the networking community. Today the Bundle Protocol (RFC 5050) is the standard communication protocol in DTNs and three major implementations are available. Since DTN is still a young research area, specifications as well as implementations have not yet reached the same state of maturity as e.g. TCP protocol stacks.

As of now no quantitative analysis of the different implementation's performance or a structured evaluation of interoperability has been undertaken. In this paper we present an interoperability check and perform an extensive quantitative performance analysis of the three main Bundle Protocol implementations.

While the overall results show that all implementations provide some baseline compatibility with each other, the stability and achieved performance under stress situations varies widely between implementations.

1 Introduction

Implementing Delay Tolerant Networking (DTN) as a way to cope with intermittently connected networks has gained a lot of interest in the research community lately. The standard DTN protocol is the Bundle Protocol defined in RFC 5050 [12]. Today several Bundle Protocol implementations exist, focusing on different use cases.

As of now, no quantitative analysis of the different implementations' performance or a structured evaluation of interoperability has been undertaken. In this work we measure and compare the performance of three open-source Bundle Protocol implementations for Linux systems: DTN2, IBR-DTN and ION. We also report compatibility and interoperability results between the different implementations.

When implementing a new protocol stack for data transmission, the possible throughput is one of the most important quantities to determine. Especially in DTNs, with short contacts between the nodes, the utilization of the available bandwidth determines the efficiency of the whole network. The performance of DTN setups is often analysed only theoretically through the use of specialised

simulators such as The ONE [7]. As these simulators only use approximations of the Bundle Protocol, the observed performance might not be in line with the performance achieved by real DTN implementations in a deployed application.

As of now, the performance of the three main Bundle Protocol implementations has only been measured punctually and due to different setups the results are not directly comparable. There has not been a comprehensive comparison of the three implementations with a well defined experimental setup. In this paper, we systematically investigate the performance over the omnipresent TCP convergence layer.

The remainder of this document is organized as follows: In section 2 we present related work. Section 3 introduces the three Bundle Protocol implementations and sheds some light on the specialities of each implementation relevant to our experiments. In section 4 we evaluate the network performance of the implementations and examine the interoperability when using the implementations in heterogeneous environments. Finally, in section 5 we summarize our findings and conclude.

2 Related Work

E. Oliver and H. Falaki [9] have built a testbed for DTN2 evaluation on low-power, low-cost computers. Their performance analysis considers wired and wireless links between four nodes in the experiments and discloses a correlation between the size of a bundle and the possible throughput.

A short comparison of the throughput of ION to the DTN2 implementation is done in [1]. The authors report bandwidths of over 800 MBit/s with a RAM based setup, but omit many important details of the measurement setup.

In [11] IBR-DTN is introduced and some basic tests are presented comparing the performance of the IBR-DTN implementation to DTN2.

The Jet Propulsion Laboratory evaluated the readiness of ION for space missions in the DINET Experiment [13]. During a period of 27 days, some 300 images were transmitted from JPL nodes located on the earth to a spacecraft in the orbit. Then they were automatically forwarded from the spacecraft back to the JPL nodes. This experiment was exercising DTN's bundle origination, transmission, acquisition, dynamic route computation, congestion control, prioritization, custody transfer, and automatic retransmission procedures, both on the spacecraft and on the ground.

3 Bundle Protocol Implementations

This work evaluates the three open-source Bundle Protocol implementations for Linux sytems: DTN2, IBR-DTN and ION. Since these implementations focus on different applications, certain design decisions for routing, storage or API differ widely. The following section briefly introduces the implementations and outlines notable design features and highlight some of the characteristics of each daemon which have been proven relevant to our evaluation.

In this paper we use the term "disk" storage to refer to bundles being stored persistently on a medium such as a hard drive or flash storage while we refer to storage relying on volatile system memory such as RAM as "memory" storage

3.1 DTN2

DTN2 [3] is the reference implementation of the Bundle Protocol by the Delay Tolerant Networking Research Group (DTNRG). It provides a flexible framework for DTN related experiments and can be configured and managed by a TCL console and configuration files. Extensions for routing, storage and convergence layers are easily attachable through XML interfaces. DTN2 features two different built-in modules for bundle storage: A memory based storage and a disk based storage relying on the Berkeley DB library.

3.2 IBR-DTN

IBR-DTN [11] is a lightweight, modular and highly portable Bundle Protocol implementation designed for embedded systems. The included DTN daemon provides different routing schemes as well as a discovery mechanism.

The IBR-DTN implementation has several flavours of disk storage. Since it has to deal with potentially large blocks as part of a bundle, a BLOB management unit provides a transparent access to container objects that store the data in memory or on disk based on the runtime configuration. These containers are used to store all potentially large blocks (e.g. payload blocks) and avoid the usage of RAM for raw data, if configured properly. Bundle objects composed of several blocks are stored in a bundle storage module for later usage. By default all bundles are held non-persistent in data structures and depending on the BLOB configuration raw data is stored on disk or in memory. If the storage is configured to use disk storage for bundles, all bundle objects are serialized into files and the corresponding object is destroyed, freeing the RAM occupied by this bundle until it is needed again. In this case the store mechanism is persistent and survives reboots or power-fails. As an alternative, IBR-DTN also offers a disk-based storage that is based on the SQLite library.

3.3 ION

Interplanetary Overlay Network (ION) [1] is a Bundle Protocol implementation from the Jet Propulsion Laboratory (JPL) specifically developed to run in robotic spacecrafts on a Real-time operating systems (RTOS). However, besides VxWorks, it also runs in x86 Linux distributions.

ION is designed much like a database and the bundle storage is based on the Simple Data Recorder (SDR), a component that already runs in spacecrafts. It can be configured to store data on disk, in memory or to use both media. The SDR supports a transaction mechanism that ensures database integrity in case of failing database operations. ION allows configuring, which combination of storage mechanisms should be used. This also allows using disk and memory storage at the time, although the outcome is not documented.

The whole system is based on a shared-memory concept that is also used to communicate with sender and receiver processes. Instead of supporting discovery of neighbors, it relies on scheduled wireless contacts. It is optimized for links with small available bandwidths and supports Compressed Bundle Header Encoding [2] (CBHE) for the Bundle Protocol Headers as well as the Licklider Transmission Protocol [10] (LTP). The authors state, that the main purpose of ION is to use CBHE and the LTP, whereas this paper focuses on standard Bundle Protocol with TCPCL to enable comparability and interoperability with DTN2 and IBR-DTN.

4 Evaluation

In networks with intermittent connections the throughput is especially important, as the goal is to take as much benefit from transient connections as possible. In this evaluation we have concentrated on the throughput that can be achieved in different scenarios. We have evaluated the raw throughput using a GBit LAN connection, the throughput in a bandwidth-limited opportunistic scenario, the possible performance that can be achieved when using different implementations as sender and receiver and finally the effect of the TCPCL segment length onto the throughput.

4.1 Experimental Setup

All experiments were conducted in a controlled environment. As target for all implementations we used computers equipped with an Athlon II X4 IV 2.8 GHz including 4 GiB RAM running Ubuntu Linux. The computers are equipped with a Samsung F3 500GB (HD502HJ) hard drive connected to the on-board SATA controller. The computers are connected using 1 GBit Ethernet. For the basic throughput tests we did not simulate bandwidth-constrained links or disruption, as our goal was to test the raw performance of DTN implementations rather than their functionality in disrupted networks.

On the PCs we used different physical Ethernet ports for the Bundle Protocol traffic and the traffic used for controlling and monitoring the experiments. The GBit NICs for Bundle Protocol traffic used a Realtek (RTL8111 / 8168B) controller. The raw TCP throughput achieved by this setup is ∼940 MBit/s which has been measured using iperf[1]. In this work we used DTN2 version 2.7, IBR-DTN version 0.6.3 and ION version 2.4.0. All implementations have been measured using their respective default configuration unless otherwise noted.

For ION, we have removed the artificial rate limit for the TCPCL that otherwise prevents ION from exceeding 90 MBit/s. To prevent missing bundles, we had to enable reversible transactions in IONs storage system.

4.2 API and bundle storage performance

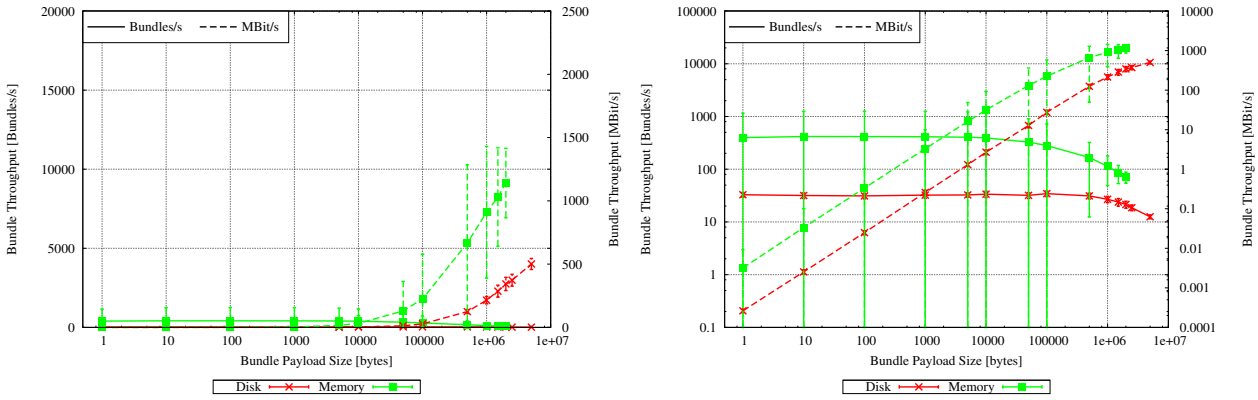
This experiment focuses on API and bundle storage performance and did not involve any network transfers. While in classical networking protocols there exists a code path from the application to the hardware driver with probably only simple and small ring buffers between the layers, a DTN implementation has to provide some form of (semi)permanent storage, which keeps and manages bundle data and accompanying meta information. Therefore, submitting packets from the application layer to the network is a more heavyweight operation in DTN protocol implementations compared to other networking protocols. It is clear that the speed at which a DTN implementation can put bundles into the storage or retrieve them will also fundamentally limit the maximum bandwidth the daemon can sustain under ideal conditions.

On each node we measured the time to store and receive 1000 bundles of varying payload size. For each payload size we performed 10 runs. The plots show the arithmetic mean bandwidth and bundle frequency of all ten runs including the standard deviation. The daemons have been restarted after performing the 10 runs for each payload size, to prevent that old runs can influence the following measurements. For DTN2 we used `dtnsend` for sending and `dtncat` for retrieving bundles. For IBR-DTN we use `dtnsend` and `dtnrecv`. For ION we use `bpsendfile` and `bprecvfile`. The tools

[1] `http://iperf.sourceforge.net/`

were used (IBR-DTN) or modified (ION, DTN2) in such a way, that a single call was sufficient to create or retrieve the 1000 bundles en-bloc so we do not measure the overhead of starting the tools over and over again.

For each implementation we measured the different available storage backends. This experiment gives a good upper bound on the performance an implementation can reach: A DTN implementation can not sustain linespeeds larger than its maximum performance receiving new data locally through its API. While for "common" networking protocols such as TCP the API (bsdsockets) normally is only a small layer of code which ties directly to the kernel via syscalls, and thus is only speed limited by the hardware, all tested implementations in this work are pure user space implementations. IBR-DTN and DTN2 use a socket based API to connect applications to the daemon. IBR-DTN can use Unix socket instead of a TCP socket when operating locally, but both approaches are more costly than using a syscall interface. ION uses a shared memory approach to facilitate IPC between applications and the core components. Independent from the API all implementations have to store received bundles in some kind of storage, which means that API performance can be IO-bound. This is different from protocols such as TCP, where only a relatively small amount of in-flight data is cached and applications are blocked when the internal buffers are filled.



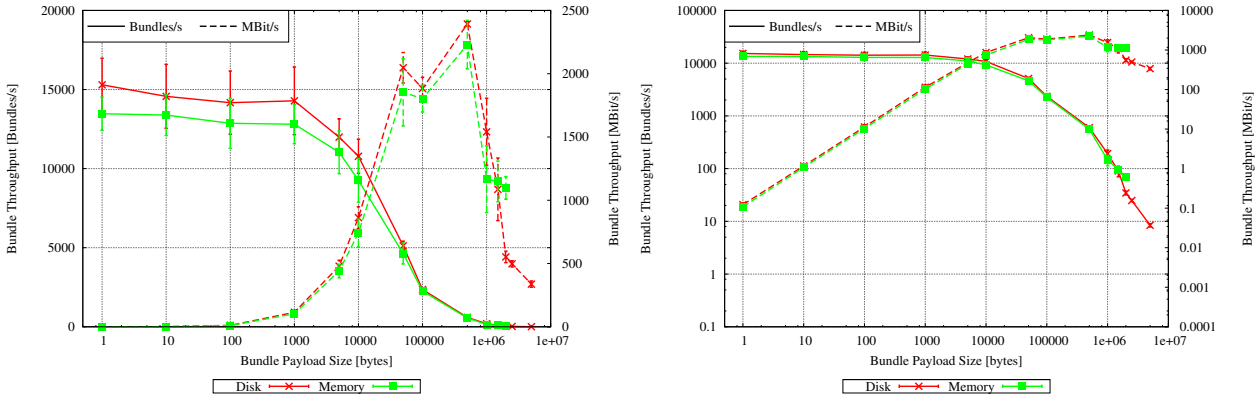
(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 1: DTN2 API send performance

The results of these tests can be seen in Figures 1, and 2 (DTN2), in Figures 3 and 4 (IBR-DTN), and in Figures 5 and 6 (ION). The logarithmic x-axis shows the size of the bundle payload. Solid lines indicate the number of bundles processed per second, dashed lines indicate the throughput to or from the daemon in MBit/s. Each figure is presented twice, either with linear or logarithmic y-axis.

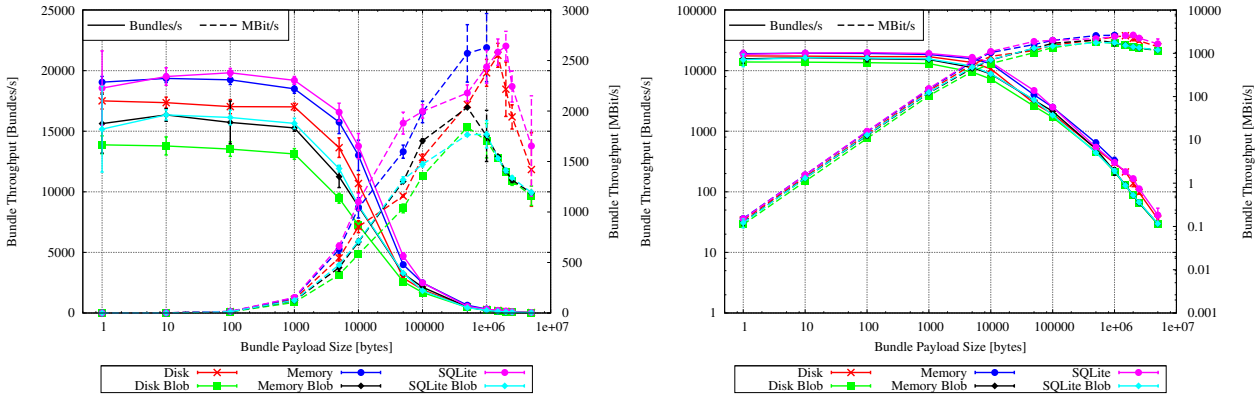
For the send case it can be seen, that for small bundle sizes all implementations are limited by the bundle frequency, i.e. the overhead for processing the individual bundles. Figure 1 shows that DTN2 reaches a bundle frequency of around 407 bundles/s for bundles $\leq 10\,000$ bytes and memory storage. For larger bundles, the frequency decreases as expected since now the tests are limited by the bandwidth provided by the storage backend. DTN2 reaches a maximum send throughput of 1140.1 MBit/s for memory based storage with disk storage being gradually slower. In Figure 3, IBR-DTN shows a similar behaviour whereas the bundle frequency starts to decrease for bundles ≥ 1000 bytes. However, the overall frequency is significantly higher for smaller bundles while the frequency for larger bundles is almost the same because the bandwidth of the hard drive or memory



(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 2: DTN2 API receive performance

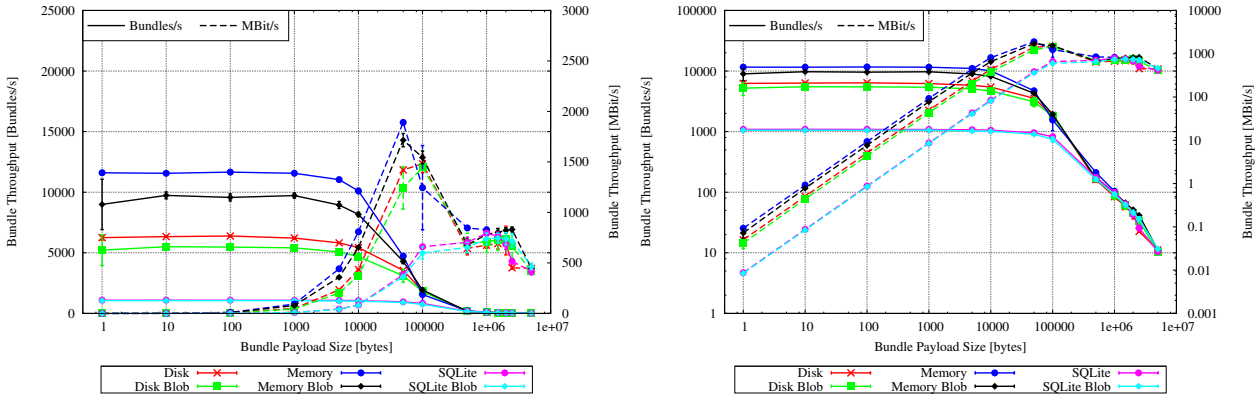
is the limiting factor. IBR-DTN shows in interesting behaviour since the throughput reaches its maximum for bundles of 2 000 000 bytes. This is caused by disk caching mechanisms and is a side effect of the 4 GiB of RAM in our test machines. ION's bundle send frequency in Figure 7 is comparable to DTN2 for smaller bundle sizes. However, ION can sustain the frequency even for larger bundles and consequently reaches a throughput of more than 7000 MBit/s. All in all, IBR-DTN has the highest bundle frequency for smaller bundle sizes while ION reaches the highest throughput for larger bundles.



(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 3: IBR-DTN API send performance

The receive test shows comparable results: DTN2 (fig. 2) and IBR-DTN (fig. 4) are bundle frequency limited for small bundles. However in contrast to the sending case here DTN2 achieves significantly higher bundle frequencies compared to IBR-DTN. In both cases throughput raises, reaches a maximum and falls back to lower speeds. The initial increase followed by a decrease in throughput can again be explained by caching: For smaller bundle sizes, all received bundles fit into the file system cache and can thus be retrieved very fast. For larger amount of data performance is again limited by available disk bandwidth. This can be seen by the IBR-DTN memory storage

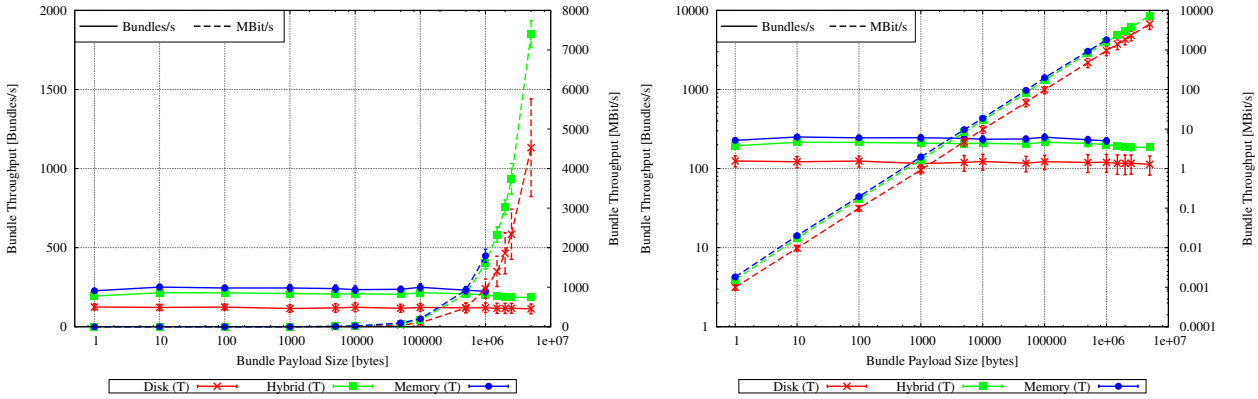


(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 4: IBR-DTN API receive performance

performance that does not write anything to disk and thus does not fall back to disk bandwidth. ION (fig. 6) again has a significantly lower bundle frequency with an interesting behaviour. The frequency is the highest for bundles between 1000 bytes and 100 000 bytes whereas it is lower for smaller and larger bundles. For the receive case, ION is not able to sustain the bundle frequency for larger bundles and reaches a maximum throughput of slightly above 1000 MBit/s.



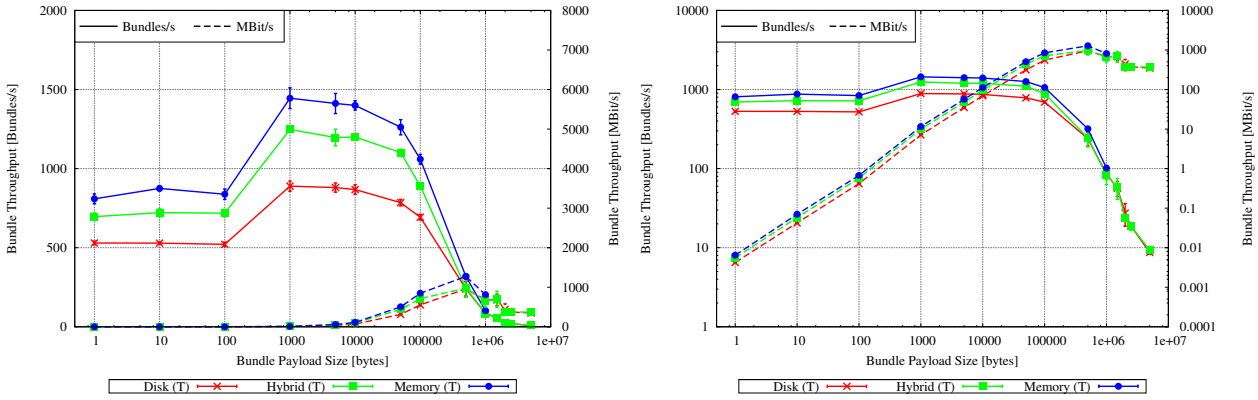
(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 5: ION API send performance

A direct comparison of the send and retrieve performance of the disk based storages is given in fig. 7 and fig. 8. Furthermore, figures 9 and 10 show a comparison of the API send and receive performance for memory based storages.

In summary it can be seen that when dealing with large amounts of data the underlying storage limits the bandwidth a DTN daemon can sustain over a period of time. A low bundle processing overhead is important to deal efficiently with small bundle sizes and a low amount of total data. For the storing case IBR-DTN outperforms DTN2 and ION in this test, for the retrieve case it is vice versa. However, since the bundle frequency DTN2 achieves when storing bundles is lower than IBR-DTN's bundle frequency upon retrieving, it is to be expected that in a situation where bundles

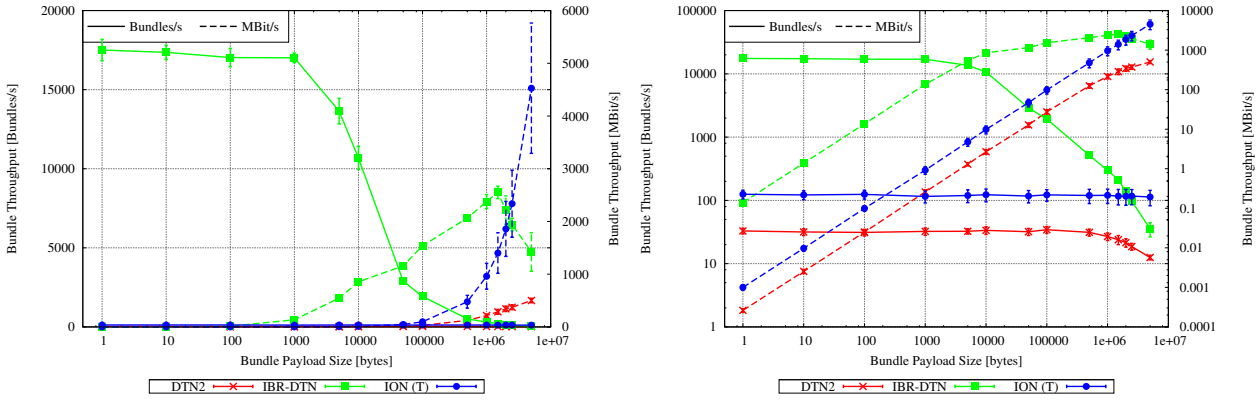


(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 6: ION API receive performance

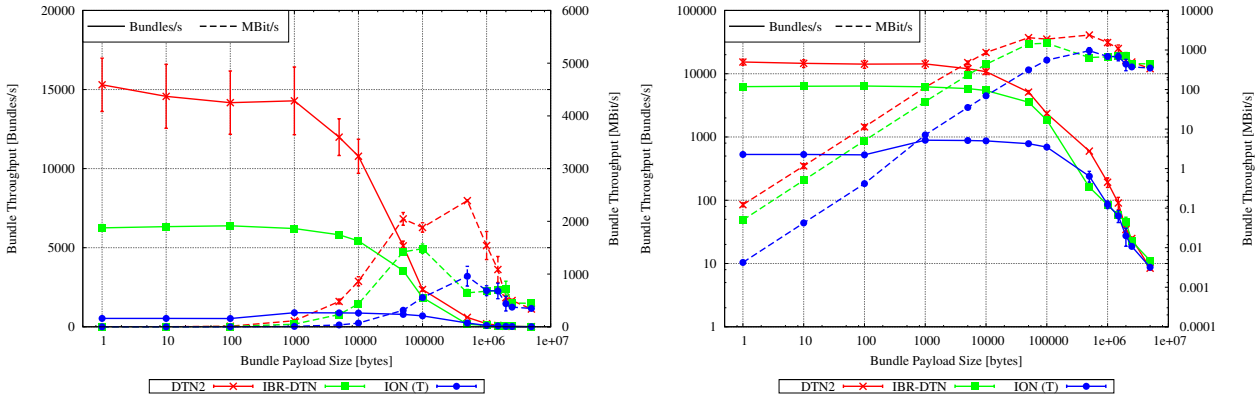
are continuously being generated, transmitted and consumed a IBR-DTN setup should outperform a DTN2 setup, while a combination of an IBR-DTN sender and an DTN2 receiver might reach even higher performance. This is based on the assumption that both implementations perform equivalently well with regard to performance and efficiency when transmitting bundles. We will look into the network throughput in sec. 4.3. We analysed different combinations of daemons in our interoperability tests in sec. 4.8.2.

The extremely high bandwidth achieved by ION in the sending case can be explained by the ION architecture that uses a shared memory approach for communication between different parts of the implementation. We assume that ION has not touched the disk for the Memory and Hybrid storages, as the achieved throughput by the hybrid storage for large bundles is well above even the streaming capability of the used hard drive.



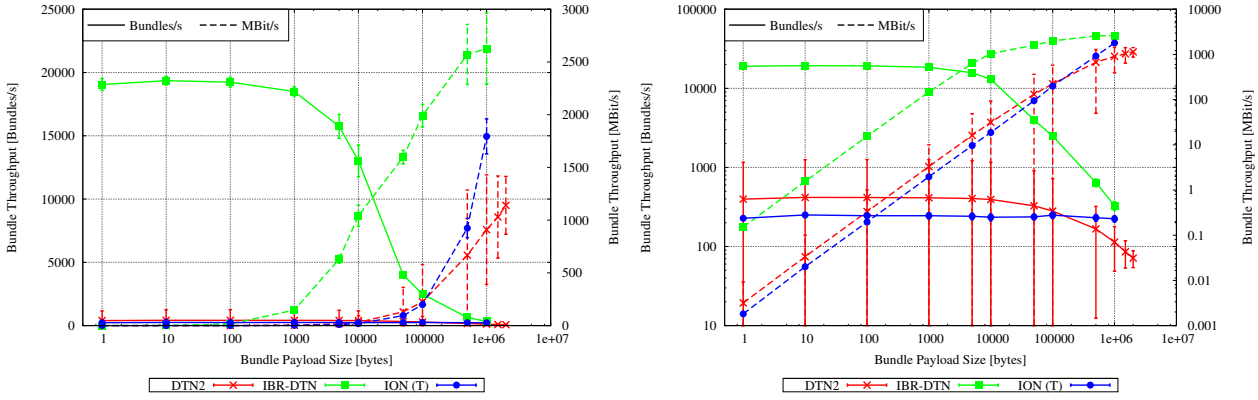
(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 7: API send performance comparison with disk storage



(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 8: API receive performance comparison with disk storage



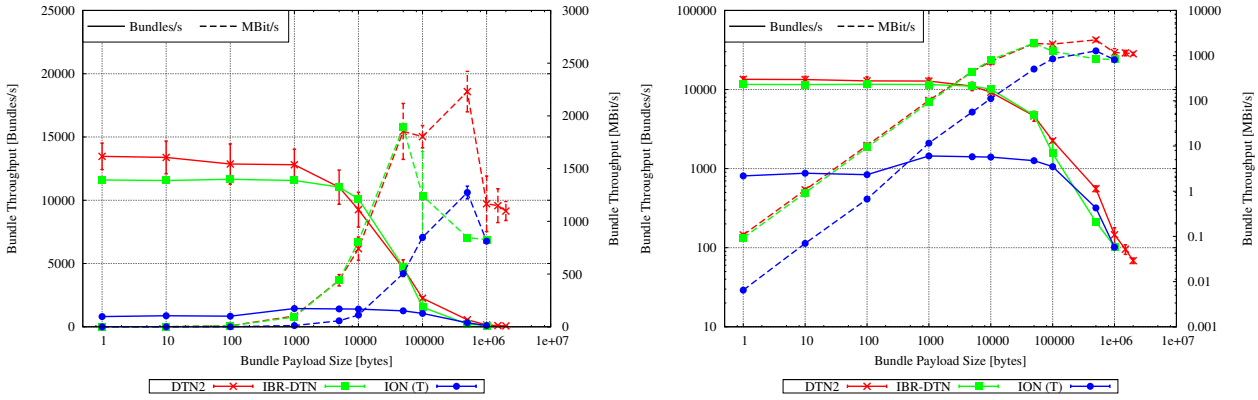
(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 9: API send performance comparison with memory storage

4.3 Network Throughput

In this experiment we measured the network performance of the different implementations. We measured throughput between two nodes that are connected via GBit Ethernet. While a GBit link might be uncommon for typical DTN applications it shows the daemons ability to saturate a given link. Failing to reach high bandwidths in this experiment indicates that bundle processing overhead in a given daemon might be too high. This will not only pose a problem with a high bandwidth link, but it could also preclude a resource-constrained node to saturate a lower bandwidth link.

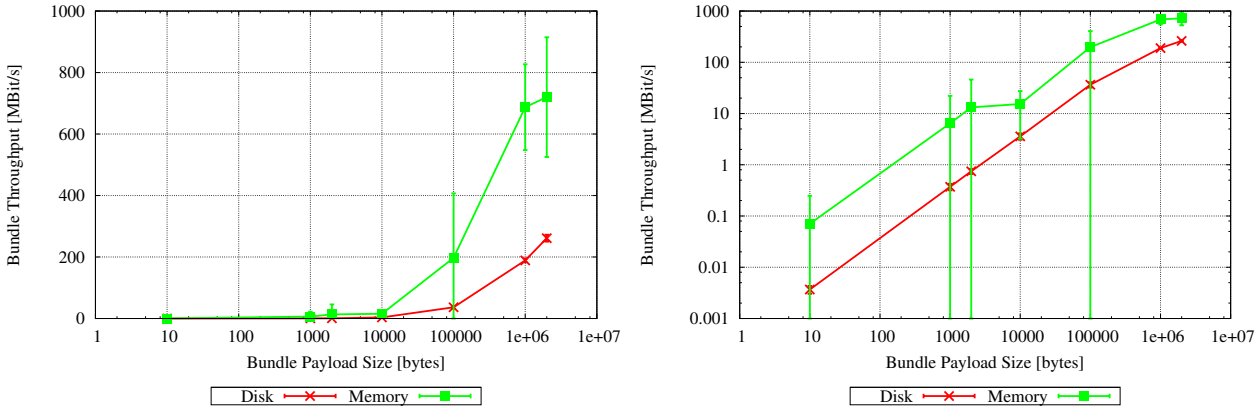
On the sender node we injected 1000 bundles for the receiver. After the bundles had been received by the sending daemon we opened the connection and measured the time until all bundles have arrived at the receiver. For each payload size we performed 10 runs. The plots in Figures 11, 12 and 13 show the average of all ten runs as well as the standard deviation. The bandwidth plotted is application layer bandwidth, i.e. it only considers the size of the payload, not protocol overhead. The daemons have been restarted after performing the 10 runs for each payload size, to prevent any influence that old runs might have on the following measurements. This experiment uses the TCP



(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 10: API receive performance comparison with memory storage

Convergence Layer.



(a) Linear Y-axis (b) Logarithmic Y-axis

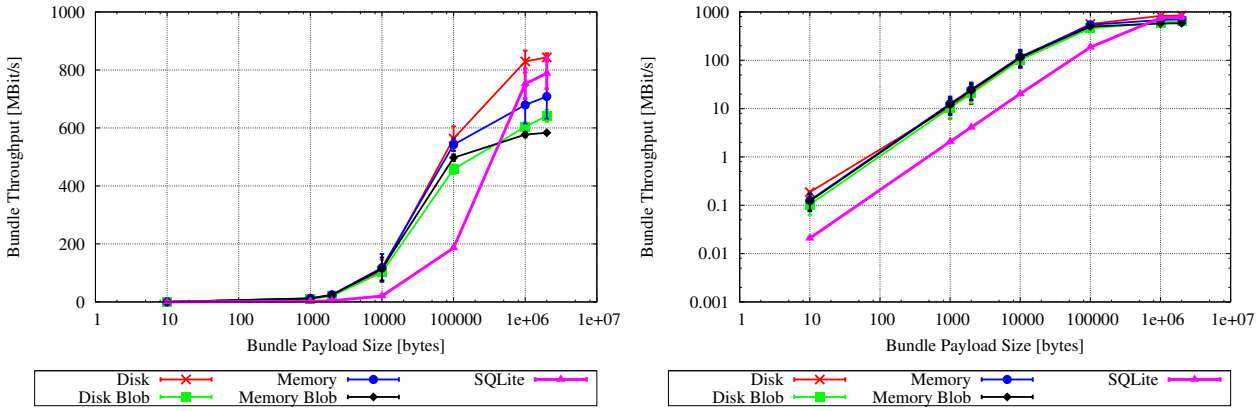
Figure 11: DTN2 network throughput

It can be seen, that IBR-DTN comes close to the theoretical limit of the link (940 MBit/s) for large bundle sizes with a throughput of up to 843.341 MBit/s for disk storage. DTN2 reaches a maximum of 719.977 MBit/s with memory storage and ION falls short with 448.833 MBit/s.

In fact all storages perform very similar for ION, which indicates that the responsible bottleneck is not located in a specific storage module.

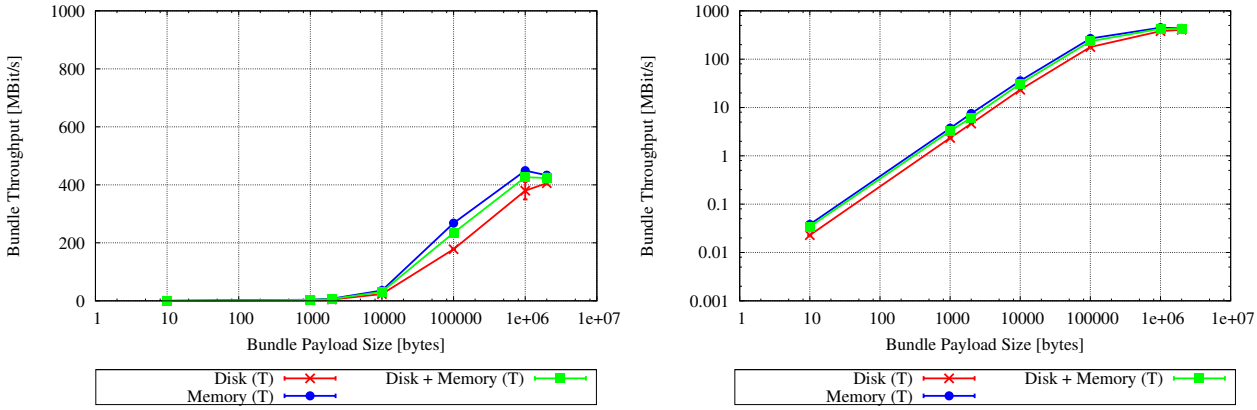
For small bundle sizes DTN2's (fig. 11) disk and memory storage achieve almost the same throughput, which indicates that the throughput is bounded by processing overhead. For bundles $\geq$ 10 kByte DTN2's disk storage achieves lower performance than memory storage, which indicates that for these sizes the throughput of the storage engine limits performance. The variances for DTN2's memory storage are extremely high. Upon further investigation, we discovered that after each run using memory storage DTN2 gets gradually slower. More details on this issue can be found in sec. 4.5.

IBR-DTN (fig. 12) shows a similar behaviour where the throughput increases with larger bundle



(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 12: IBR-DTN network throughput



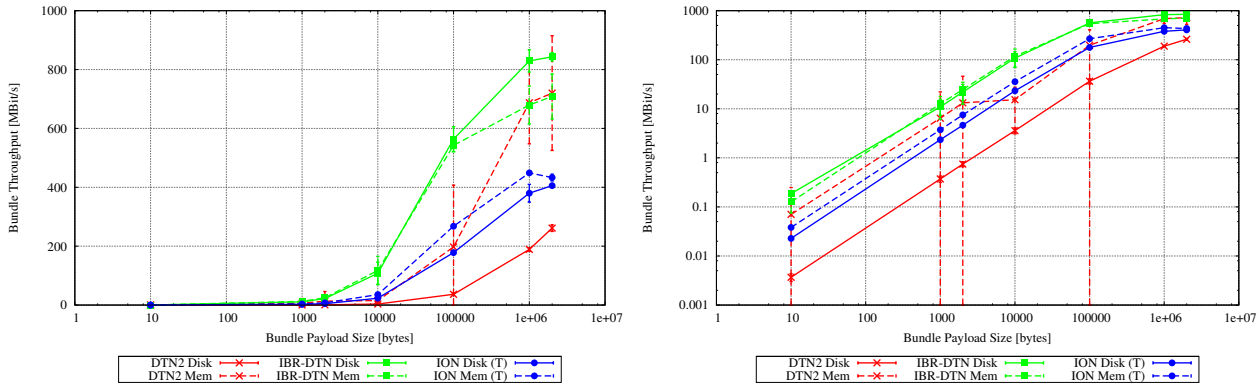
(a) Linear Y-axis (b) Logarithmic Y-axis

Figure 13: ION network throughput (with transactions)

sizes, however in absolute terms throughput is always significantly higher than DTN2's. In IBR-DTN the performance is more consistent between different storage backends, while DTN2 performance is much higher when using the memory-based storage compared to its disk based storage. Interestingly, IBR-DTN shows higher throughputs for disk-based storage compared to memory storage. After further investigation this turned out to be caused by the respective implementation of the storage modules. Disk-based storage uses multiple threads and makes good use of the multiple CPU cores in our test machines while memory-based storage only uses one thread and subsequently only one core. The SQLite storage of IBR-DTN is the slowest for smaller bundles but then picks up and is only gradually slower than the disk-based storage for bundles $\geq 1\,000\,000$ bytes.

ION (fig. 13) shows a performance between the two storage backends of DTN2. It is noteworthy, that the two ION storage backends have comparable performance for all bundle sizes. This fact implies that the bottleneck in ION is not within the storage module but in the processing of the daemon. While DTN2's memory-based storage is significantly faster than ION, DTN2's disk-based storage is also significantly slower than ION.

A comparison of the throughput of all disk based storages can be seen in fig. 14. Clearly in the network throughput test IBR-DTN is the winner, while the picture between DTN2 and ION is not so clear.

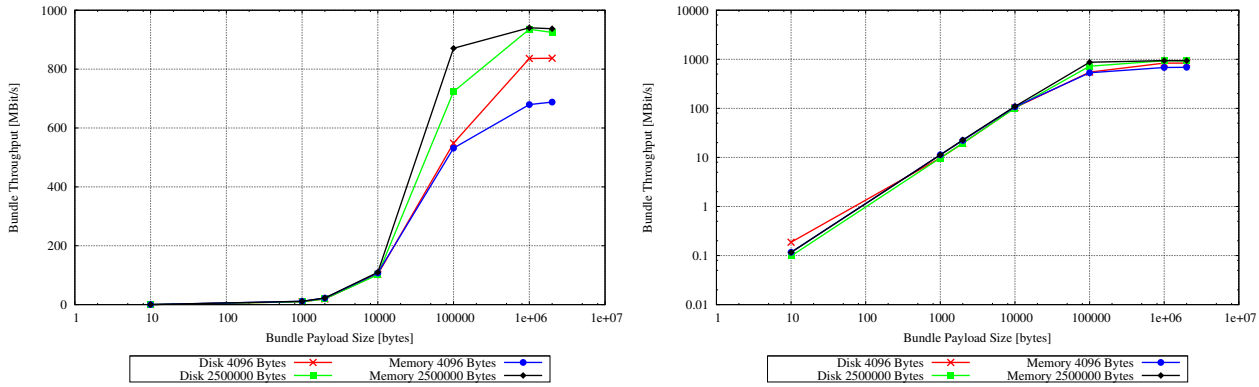


(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 14: Network throughput comparison

4.4 TCPCL Segment Length



(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 15: Impact of TCPCL segment length onto Throughput for IBR-DTN

As seen in section 4.3 neither implementation was able to saturate the maximum bandwidth of the link. Apart from performance limits in the daemons, unsuitable parameters used in the Bundle Protocol can have an effect on throughput. As DTNs are mostly deployed in wireless networks it can be assumed that the implementations have been tested in such setups, where the available bandwidth is much lower than in our test cases.

The parameter which is most likely to influence throughput is the TCP Convergence Layer (TCPCL)[4] segment length. Essentially, a larger segment length means less processing overhead: Executing fewer TCP send syscalls saves processing overhead in the daemon and avoids unnecessary

switches between userspace and the kernel. Also, since every segment is acknowledged by the receiver, for larger segment lengths there is less overhead processing the TCPCL acknowledgements.

The TCPCL segment length defines how many bytes the TCPCL will transmit as a single block. This defines the amount of data that is acknowledged by a single Ack and also influences the granularity of reactive fragmentation. It is to be expected that by using a larger TCPCL segment size a daemon can saturate a higher bandwidth. Both DTN2 and IBR-DTN use 4096 kBytes as default segment length, while ION does not segment bundles at all. To show the influence of the TCPCL segment length, we have measured the achievable throughput with the default value of 4096 bytes and an extreme value of 2.5 MB.

Figure 15 shows the influence of the segment length for IBR-DTN. It can be seen, that for bundles larger than the default segment length of 4096 bytes the performance is increased significantly. The fastest measured throughput was for memory-storage with the increased segment length of 2.5 MB and a bundle size of 1 MB. In this setting, IBR-DTN reaches a speed of 935.200 MBit/s which is 99.489 % of the theoretical maximum of the link.

This measurement has clearly shown that tweaking the TCPCL segment length can significantly increase the throughput of a Bundle Protocol implementation.

4.5 DTN2 Throughput Variances

As shown in sec. 4.3, DTN2 exhibits high variances between different runs of the same bundle size and storage, especially for medium-sized bundles when using memory storage. Figure 16 shows the throughput over different measurement runs. Especially for the smaller bundle size a clear trend of decreasing speed in each measurement run can be seen. For the big bundle sizes the behaviour gets more erratic.

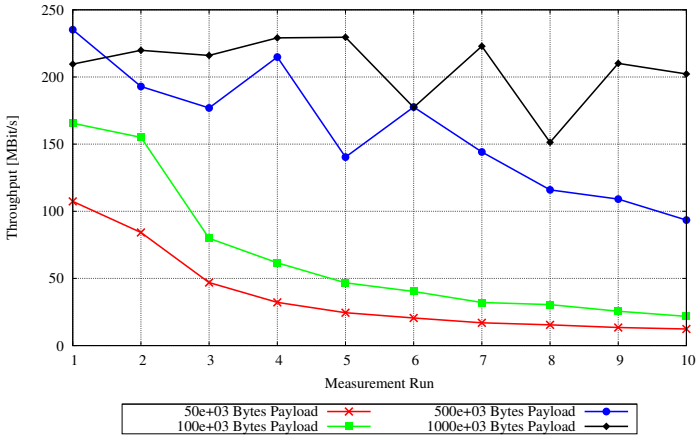
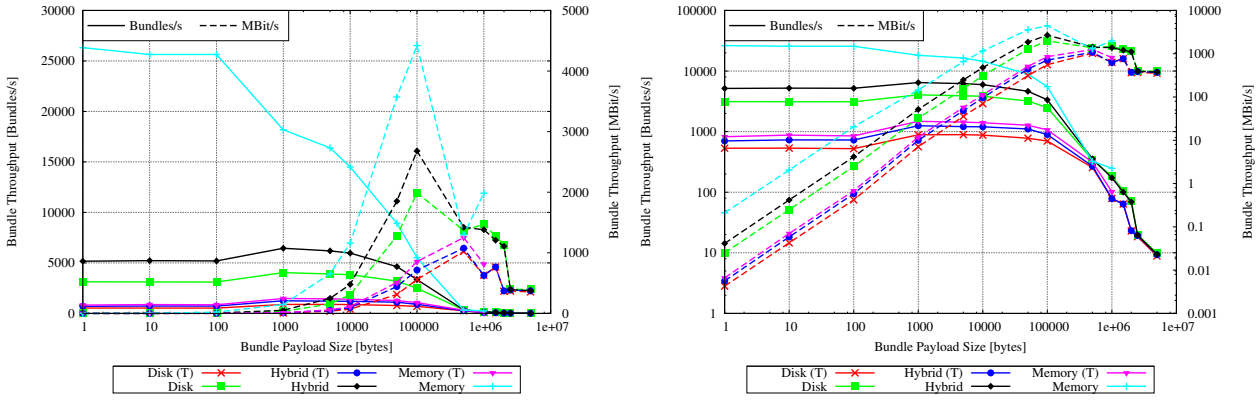


Figure 16: DTN2 throughput with different payload sizes over multiple measurement runs.

This behaviour might indicate some resource management problems within DTN2, i.e. some resources might not be freed when not needed anymore. As we restarted all daemons for each new set of parameters, the results presented before are not influenced by the order of the experiments.



(a) Linear Y-axis

(b) Logarithmic Y-axis

Figure 17: ION comparison of the API performance for different storage configurations.

4.6 ION Storage Variants

ION allows configuring the underlying storage using a number of flags that enable or disable storage types. This enables the user to use multiple storage options at the same time. Figure 17 shows the performance of the ION API with different storage configurations and bundle sizes. It can be seen, that memory storage has the highest throughput in bundles and bytes with significant benefits for bundles sizes below 1000 bytes. Also, the combination of disk and memory storage (denoted as hybrid) is significantly faster than normal disk storage. When we have enabled reversible transactions, the performance drops at least an order of magnitude, but still memory storage is faster than hybrid that is still faster than disk storage.

Although this high performance is promising, we were unable to run our throughput tests without reversible transactions enabled, as we experienced severe bundle loss without any error messages under these conditions. Therefore, for our experiments we had to resort to the slower alternative.

4.7 Opportunistic Throughput

The goal of the scenario in sec. 4.3 was a clean throughput test to reveal the maximum performance of each implementation. In this second scenario the goal is to get realistic throughput values in a network with opportunistic contacts. Based on a scenario for public transport systems published in [8], we have two fixed nodes which are out of range of each other and a mobile node that commutes between them. The timing is based on measurements published in [5]. In each cycle, the mobile node first makes contact with the sender for 72 s, then has no contact for 30 s to make contact with the receiver for 72 s afterwards. After another 30 s without contact, the cycle starts again with the whole experiment lasting 30 minutes. The contacts are simulated using iptables. We limited the TCP throughput to 12.7 MBit/s based on the results presented in [5] using a Token Bucket Filter (tbf) based on the Linux Classless Queuing Disciplines (qdisc). The bundle creation speed is fixed and ensures that always enough bundles for the next contact are present on the sender node. Under optimal conditions this setup would allow transmitting a total of 1000.59 MByte of raw TCP data in 30 minutes.

The hardware used is the same as described in section 4.1. Each experiment is repeated five times for different bundle sizes with DTN2 and IBR-DTN using the memory storage backends. ION

is not able to compete in this test, since it does not support opportunistic contacts. We used flooding as routing module for DTN2 and epidemic routing for IBR-DTN, since they are virtually identical in functionality. The total amount of payload that could be transferred is listed in table 1, while the values represent the average and the standard deviation of 5 measurement runs.

	250 kB	**500 kB**	**1 MB**	**2 MB**
DTN2	918.2 MB ± 4.18 MB	917.3 MB ± 2.28 MB	917.6 MB ± 2.30 MB	908 MB ± 6.63 MB
IBR-DTN	841.3 MB ± 2.96 MB	913.1 MB ± 0.65 MB	912.6 MB ± 1.52 MB	902 ± 0.00 MB

Table 1: Transmitted data and standard deviation in the opportunistic scenario

The results show, that in these tests DTN2 slightly outperforms IBR-DTN, with the difference being more pronounced for smaller bundle sizes. The slightly lower performance of IBR-DTN was expected, since the memory-based storage produces a lower throughput as already stated in Figure 12. Also, IBR-DTN has a higher processing overhead when the storage subsystem keeps track of a high number of bundles. Another influencing factor is the fact that DTN2 supports reactive bundle fragmentation that is helpful whenever a bundle has been partially transferred. IBR-DTN does not yet support this feature, so that incomplete bundles are dropped and have to be completely retransmitted upon the next contact.

4.8 Interoperability

Interoperability is an important issue as the DTN community is rather small and the Bundle Protocol quite young. Not only can incompatibilities expose bugs in the respective daemons but also more importantly they might also highlight ambiguities or undefined areas in the Bundle Protocol specification. For the interoperability tests we check whether the daemons can discover each other, which is important for applications relying on opportunistic contacts. However, the most important thing is Bundle Protocol compatibility: If all daemons adhere to the Bundle Protocol [12] and TCPCL [4] specification they should be able to exchange bundles. For the working combinations we will measure the data throughput to give an estimate of the performance that can be expected in heterogeneous environments.

4.8.1 Discovery

Typically, discovery mechanisms are used to detect an available link between two daemons. The IPND draft [6] is a specification for such a mechanism. However, DTN2 implements a proprietary IP-Discovery mechanism using TCP or UDP announcements for IPv4 address/port pairs while IBR-DTN supports IPND in versions 0 and 1 in addition to the DTN2 IP-Discovery mechanism. The ION implementation is strictly focussed on scheduled contacts and does not possess a discovery mechanism.

To prove the interoperability between the three implementations, we used a setup with the standard TCP convergence layer and did a simple test by forwarding a single bundle from one implementation to another using default settings. Before forwarding the bundles, the daemons had to discover each other and setup a TCP connection. We have found out that DTN2 and IBR-DTN

are basically compliant and can dynamically discover each other. However, configuration is necessary to specify the common discovery mechanism and UDP port to use.

4.8.2 Throughput

TX \ RX	DTN2	IBR-DTN	ION
DTN2	197.231 MBit/s ±210.016 MBit/s	193.202 MBit/s ±198.240 MBit/s	71.574 MBit/s ±9.456 MBit/s
IBR-DTN	677.153 MBit/s ±173.756 MBit/s	542.337 MBit/s ±21.533 MBit/s	76.161 MBit/s ±0.623 MBit/s
ION	454.513 MBit/s ±53.763 MBit/s	420.743 MBit/s ±11.640 MBit/s	267.832 MBit/s ±2.028 MBit/s

(a) 100 KBytes payload size

TX \ RX	DTN2	IBR-DTN	ION
DTN2	687.329 MBit/s ±139.590 MBit/s	635.088 MBit/s ±49.902 MBit/s	93.116 MBit/s ±0.568 MBit/s
IBR-DTN	881.463 MBit/s ±72.149 MBit/s	679.45 MBit/s ±64.362 MBit/s	89.915 MBit/s ±4.561 MBit/s
ION	871.677 MBit/s ±76.428 MBit/s	926.005 MBit/s ±4.009 MBit/s	448.833 MBit/s ±3.592 MBit/s

(b) 1 MBytes payload size

Table 2: Average Interoperability Throughputs using memory-based Storage

To determine, how the interaction between different implementations impacts throughput, we conducted some additional performance tests. We created 1000 bundles on the sender node and then measured the time it takes to transfer these bundles to the receiver node. We used the TCPCL in its respective default configuration and opted to use memory storage to preclude any performance impacts due to disk caching and similar issues. We set up static routing and opened and closed connection between daemons using either built-in methods such as a command line interface or an admin interface (ION, DTN2) or iptables (IBR-DTN). We ran these measurements for all 9 possible pairs of implementations and the results are stated in table 2a for 100 KByte payload size and in table 2b for 1 MByte payload. Each test was performed 10 times and the tables show the average throughput and the standard deviation of these runs.

The results for DTN2 show that it can send 100 KByte bundles with up to 197.231 MBit/s, while it can receive such bundles with up to 677.153 MBit/s. Also, DTN2 can send 1 MByte bundles with up to 687.329 MBit/s while it can receive them with up to 881.463 MBit/s. This allows the conclusion that DTN2 has a bottleneck in the sending components that limits the amount of bundles that can be processed per time interval.

IBR-DTN is able to send 100 KByte bundles with up to 677.153 MBit/s while it can receive such bundles with up to 542.337 MBit/s. For 1 MByte bundles, IBR-DTN can transmit with up to 881.463 MBit/s and receive with up to 926.005 MBit/s. This leads to the conclusion that IBR-DTN has a bottleneck in the transmitting component limiting the throughput of this implementation.

However, since IBR-DTN is the fastest sender and receiver in both experiments when communication with other implementations, it is surprising that IBR-DTN does not produce higher throughput with another instance of IBR-DTN. This does not allow a clear conclusion.

Finally, ION transmitted bundles of 100 KByte with up to 454.513 MBit/s while it received such bundles with up to 267.832 MBit/s. Also, ION transmitted bundles of 1 MByte with up to 926.005 MBit/s and received such bundles with up to 448.833 MBit/s. While the transmit values are competitive to DTN2 and IBR-DTN, the receive performance falls short reaching less than half the throughput of the highest measured value.

An interesting result of the interoperability throughput tests is, that for smaller payload sizes, the fastest sender and receiver combination is IBR-DTN and DTN2, while for larger bundles the fastest combination is ION and IBR-DTN. Intuitively one would have expected to find the highest performance between a sender and receiver of the same implementation. However, this confirms the result that DTN2 is restrained by a bottleneck in the sending component. It can also be concluded, that for bundle transfer between the same implementation DTN2 reaches the highest throughput for 1 MByte payload size, slightly ahead of IBR-DTN with ION far below. However, for 100 KByte bundles, IBR-DTN significantly outperforms the two competitors.

5 Conclusions

To the best of our knowledge, we have performed the first extensive study of performance and interoperability of the three relevant Bundle Protocol implementations. We have looked at the throughput that can be achieved using TCPCL as well as the effects of the storage systems available for the implementations. Further, we have investigated which performance can be achieved in an opportunistic scenario and how the different implementations interact with each other. Finally, we have evaluated the effects of the TCP segment length onto throughput.

Generally the three implementations are interoperable. The achievable performance depends on a significant number of factors and cannot be predicted easily. This becomes especially clear when looking at the result that the fastest way of transmitting bundles is to use ION as sender and IBR-DTN as receiver.

One of the major influencing factors is the underlying storage system that effectively limits the achievable throughput for links with high bandwidth. Disk storage can be considered the default option for DTNs and is not only influenced by the throughput of the disk but also by the concrete implementation and caching mechanisms of the underlying operating system. Especially when the total amount of bundles exceeds the available RAM, this can be a bottleneck.

Evaluating the performance and interoperability of Bundle Protocol implementations is important for users and developers of DTNs alike. DTN users want to choose their implementation wisely based on their usage scenario. Developers want to tune their implementations to achieve the best usability for a specific target scenario.

Acknowledgements

This work has been partially supported by EFRE (European fund for regional development) project OPTraCom (W2-800-28895) and by the NTH School for IT Ecosystems.

References

[1] S. Burleigh. Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol. In *4th IEEE Consumer Communications and Networking Conference, 2007. (CCNC 2007)*, pages 222–226, Jan. 2007.

[2] S. Burleigh. Compressed Bundle Header Encoding (CBHE). *IETF Draft*, Feb. 2011.

[3] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra. Implementing Delay Tolerant Networking. Technical report, IRB-TR-04-020, Dec. 2004.

[4] M. Demmer and J. Ott. Delay Tolerant Networking TCP Convergence Layer Protocol. *IETF Draft*, Nov. 2008.

[5] M. Doering, W.-B. Pöttner, T. Pögel, and L. Wolf. Impact of Radio Range on Contact Characteristics in Bus-based Delay Tolerant Networks. In *Eighth International Conference on Wireless On-Demand Network Systems and Services (WONS 2011)*, pages 195–202, Bardonecchia, Italy, Jan. 2011.

[6] D. Ellard and D. Brown. DTN IP Neighbor Discovery (IPND). *IETF Draft*, Mar. 2010.

[7] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10, 2009.

[8] S. Lahde, M. Doering, W.-B. Pöttner, G. Lammert, and L. Wolf. A practical analysis of communication characteristics for mobile and distributed pollution measurements on the road. *Wireless Communications and Mobile Computing*, 7(10):1209–1218, Jan. 2007.

[9] E. Oliver and H. Falaki. Performance Evaluation and Analysis of Delay Tolerant Networking. In *Proceedings of the 1st International Workshop on System Evaluation for Mobile Platforms*, MobiEval '07, pages 1–6, New York, NY, USA, 2007. ACM.

[10] M. Ramadas, S. Burleigh, and S. Farrell. Licklider Transmission Protocol - Specification. RFC 5326 (Experimental), Sept. 2008.

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

[12] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), 2007.

[13] J. Wyatt, S. Burleigh, R. Jones, L. Torgerson, and S. Wissler. Disruption Tolerant Networking Flight Validation Experiment on NASA's EPOXI Mission. *International Conference on Advances in Satellite and Space Communications*, pages 187–196, 2009.

Technische Universität Braunschweig
Informatik-Berichte ab Nr. 2008-08

2008-08	B. Rosic	A Review of the Computational Stochastic Elastoplasticity
2008-09	B. N. Khoromskij, A. Litvinenko, H. G. Matthies	Application of Hierarchical Matrices for Computing the Karhunen-Loeve Expansion
2008-10	M. V. Cengarle, H. Grönniger B. Rumpe	System Model Semantics of Statecharts
2009-01	H. Giese, M. Huhn, U. Nickel, B. Schätz (Herausgeber)	Tagungsband des Dagstuhl-Workshops MBEES: Modellbasierte Entwicklung eingebetteter Systeme V
2009-02	D. Jürgens	Survey on Software Engineering for Scientific Applications: Reuseable Software, Grid Computing and Application
2009-03	O. Pajonk	Overview of System Identification with Focus on Inverse Modeling
2009-04	B. Sun, M. Lochau, P. Huhn, U. Goltz	Parameter Optimization of an Engine Control Unit using Genetic Algorithms
2009-05	A. Rausch, U. Goltz, G. Engels, M. Goedicke, R. Reussner	LaZuSo 2009: 1. Workshop für langlebige und zukunftsfähige Softwaresysteme 2009
2009-06	T. Müller, M. Lochau, S. Detering, F. Saust, H. Garbers, L. Märtin, T. Form, U. Goltz	Umsetzung eines modellbasierten durchgängigen Enwicklungsprozesses für AUTOSAR-Systeme mit integrierter Qualitätssicherung
2009-07	M. Huhn, C. Knieke	Semantic Foundation and Validation of Live Activity Diagrams
2010-01	A. Litvinenko and H. G. Matthies	Sparse data formats and efficient numerical methods for uncertainties quantification in numerical aerodynamics
2010-02	D. Grunwald, M. Lochau, E. Börger, U. Goltz	An Abstract State Machine Model for the Generic Java Type System
2010-03	M. Krosche, R. Niekamp	Low-Rank Approximation in Spectral Stochastic Finite Element Method with Solution Space Adaption
2011-01	L. Märtin, M. Schatalov, C. Knieke	Entwicklung und Erweiterung einer Werkzeugkette im Kontext von IT-Ökosystemen
2011-02	B. V. Rosić, A. Litvinenko, O. Pajonk, H. G. Matthies	Direct Bayesian update of polynomial chaos representations
2011-03	H. G. Matthies	White Noise Analysis for Stochastic Partial Differential Equations
2011-04	O. Pajonk, B. Rosić, A. Litvinenko, and H. G. Matthies	A Deterministic Filter for non-Gaussian Bayesian Estimation
2011-05	H. G. Matthies	A Hitchhiker's Guide to Mathematical Notation and Definitions
2011-06	R. van Glabbeek, U. Goltz, J.-W. Schicke	On Causal Semantics of Petri Nets
2011-07	H. Cichos, S. Oster, M. Lochau, A. Schürr	Extended Version of Model-based Coverage-Driven Test Suite Generation for Software Product Lines
2011-08	W.-B. Pöttner, J. Morgenroth, S. Schildt, L. Wolf	An Empirical Performance Comparison of DTN Bundle Protocol Implementations