Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems

Steffen Müller, David Bermbach, Stefan Tai
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {st.mueller, david.bermbach, stefan.tai}@kit.edu

Frank Pallas
FZI Forschungszentrum Informatik
Berlin, Germany
Email: pallas@fzi.de

***Abstract*—Cloud storage services and NoSQL systems are optimized for performance and availability. Hence, enterprise-grade features like security mechanisms are typically neglected even though there is a need for them with increased cloud adoption by enterprises. Only Transport Layer Security (TLS) is frequently supported. Furthermore, the standard Transport Layer Security (TLS) protocol offers many configuration options which are usually chosen purely based on chance.**

We argue that in cloud database systems, configuration options should be chosen based on the degree of vulnerability to attacks and security threats as well as on the performance overhead of the respective algorithms. Our contributions are a benchmarking approach for transparent analysis of the performance impact of various TLS configuration options and a custom TLS socket implementation which offers more fine-grained control over the configuration options chosen. We also use our benchmarking approach to study the performance impact of TLS in Amazon DynamoDB and Apache Cassandra.

Keywords*—*Cloud Security; Security Performance; Cloud Database Systems; NoSQL; SSL; TLS

I. Introduction

Cloud storage services and NoSQL systems[1], e.g., DynamoDB [2], Cassandra [3], HBase [4], MongoDB [5], or Voldemort [6], have been designed with a focus on performance, availability and elastic scalability. While the delivery model may vary[7], the one thing that all these systems have in common is that they typically abandon enterprise-grade features in favor of increased performance or availability. Typical examples of this are relaxed consistency [1] guarantees – based on the tradeoffs of the CAP theorem and PACELC model [2], [3], so that only Eventual Consistency [4] is offered – and security mechanisms that are either non-existent or only offered as optional features.

However, for systems running in a public cloud, the confidentiality and integrity of data in transport is an even larger challenge than for traditional on-premise database systems. In on-premise environments, everything is under the control of the same entity and the networks are protected by perimeters. In cloud database systems running in the public cloud, data packets have to be sent over unsecure public networks. Using geo-replication further aggravates this. Secure communication protocols like TLS (Transport Layer Security)[8] can help to alleviate security concerns by using suitable encryption and authentication mechanisms combined with cryptographic signatures. As security is a fundamental need in software systems, more and more cloud database systems add support for TLS and its myriad of encryption and hashing algorithms. Still, this comes with an unknown price in terms of performance.

This unknown performance impact of enabling TLS, if negative, is contrary to the original design goals of cloud database systems. We therefore believe that such a decision needs to be well-considered, based on detailed knowledge of implications and potential side effects. Up to now, no approach exists that delivers this kind of information. In this work, we hence propose a benchmarking approach to determine the performance impact of security design decisions in arbitrary cloud database systems. As a second contribution, we then use our approach to measure the performance impact of using specific combinations of encryption and hashing algorithms in Cassandra and DynamoDB.

The remainder of this work is structured as follows: In section II, we start with an overview of different types of middleware-based communication in cloud database systems, an overview of the TLS protocol and its potential performance impact, as well as a discussion of related work. Afterwards, in section III, we present our approach for benchmarking the performance impact of TLS. We also describe factors such an approach needs to be aware of. Next, in section IV, we describe the results of our extensive experiments with Amazon DynamoDB and Apache Cassandra using the tool implementation of our benchmarking approach. Finally, we end with a discussion and conclusion in section V.

II. Background and Related Work

In this section, we describe communication types in cloud database systems and give an overview of secure communication protocols. We also discuss existing approaches which try to measure the performance overhead of using SSL or TLS.

A. Communication in Cloud Database Systems

In cloud database systems, there are three main roles: The application is running on one or more machines and uses the

[1]We collectively refer to both classes of systems as cloud database systems.
[2]aws.amazon.com/de/dynamodb
[3]cassandra.apache.org
[4]hbase.apache.org
[5]mongodb.org/
[6]project-voldemort.com
[7]A hosted service in the case of cloud storage or self-hosted systems potentially running on compute services in the case of NoSQL systems

[8]Earlier versions of TLS were known as SSL (Secure Sockets Layer).

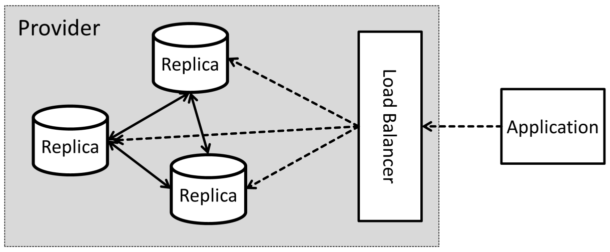

Figure 1. Schematic Overview of Communication in Cloud Database Systems

storage system as an abstraction for state management. On the provider's side, there are replica servers which actually store the data – typically replicated on several machines – and a load balancer which acts as a proxy towards the client routing his requests to the appropriate machines. Systems like HBase or Google File System [5] additionally have machines for management purposes which, for our purposes, can be treated like replica servers. In Peer-to-Peer (P2P) systems replica servers typically also include load balancer functionality. Figure 1 shows how client requests may be routed to different replica servers as well as how the latter interact.

Based on this, we can distinguish two basic types of communication in cloud database systems:

- *Application-Replica Communication* comprises the data flow from the application to the first replica server including the hop via the load balancer (dashed line in figure 1). For systems without a load balancer (which is common in P2P systems, e.g., Cassandra [6]), the extra hop via the load balancer is missing and applications send their requests directly to a replica server. Application-replica communication will often start and end in different data centers and typically uses standardized communication middleware like Thrift[9], Protocol Buffers[10], Avro[11], Web Services (REST or SOAP/WSDL), or platform-specific transport mechanisms like standard Java serialization with RMI or directly upon TCP sockets.
- *Replica-Replica Communication* happens only between replica servers (solid line in figure 1) and will only span different datacenters if geo-replication is used. While communication may also use standardized middleware solutions, frequently proprietary communication protocols build on language-specific mechanisms are used. This is often more convenient and suffices as communication is strictly confined to a closed system without the need to offer endpoints for different languages and platforms. Sometimes systems also build upon existing systems, e.g., Voldemort on top of BerkeleyDB[12] or HBase on top of Zookeeper and Hadoop Distributed File System (HDFS), and use their synchronization mechanisms.

Table I shows how several popular cloud database systems communicate internally and externally. Note, that in the case of NoSQL systems, the developer controls both types of communication and can choose various kinds of security settings. For cloud storage services, in contrast, the developer cannot affect replica-replica communication at all. Furthermore, for application-replica communication, he can only choose from the TLS options supported by the provider.

[9]thrift.apache.org

[10]code.google.com/p/protobuf

[11]avro.apache.org

[12]oracle.com/technetwork/products/berkeleydb

Table I. COMMUNICATION IN CLOUD DATABASE SYSTEMS

Storage System	**Application-Replica**	**Replica-Replica**
DynamoDB	Web services (HTTP)	?
Cassandra	Prop. Binary RPC, Thrift	Prop. Binary RPC
HBase	Prop. Binary RPC, HTTP, Protocol Buffers, etc.	Prop. Binary RPC (Zookeeper, HDFS)
MongoDB	RESTful services (HTTP)	Prop. Binary RPC
Voldemort	HTTP, Avro, etc.	Prop. Binary RPC (MySQL, BerkeleyDB)

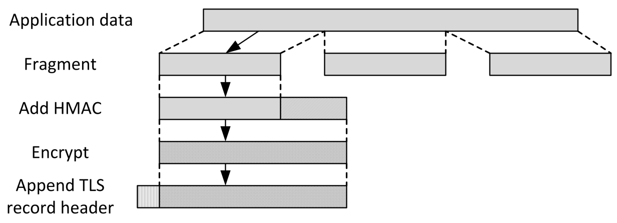

Figure 2. Composition of TLS Messages

B. Overview of Transport Layer Security

The TLS protocol has three main building blocks [7]:

1) Data packets are encrypted before sending them over the network and, thus, cannot be read by interceptors. This asserts confidentiality.
2) Before encryption, hash-based message authentication codes (HMAC) are appended to each message so that the recipient can verify the integrity of the message.
3) Certificates are used to authenticate communication partners.

During an initial handshake phase, the server first authenticates himself with his certificate; next, client and server agree on a combination of encryption and HMAC algorithms, the TLS cipher suites, as well as further session parameters[13]. Afterwards, during the bulk data transfer phase, HMACs are appended to each data packet before encrypting both data packet and HMAC. Finally, a TLS record header is added and the entire message is sent to the recipient who can then decrypt the message, verify the data integrity and reassemble the data packets. Figure 2 shows the TLS message composition during the bulk data transfer phase.

C. Performance Impact of Transport Layer Security

Generally speaking, TLS introduces a performance impact based on three root sources. First, there is a compute overhead for calculating HMACs as well as encrypting and decrypting messages. Second, the total amount of data which needs to be sent is increased which in turn affects the time necessary to

[13]Often several cipher suites are available. Typically, one cipher suite is then chosen.

transfer data. Third, the handshake phase has to be executed at least once before transmitting data – this delays the actual data transmission and, thus, increases latency. Therefore, this problem is well known and the simple performance overhead of sending data from A to B either encrypted or unencrypted is well studied:

Apostolopoulos et al. [8], Kant et al. [9], Zhao et al. [10], and Coarfa et al. [11] study the performance overhead of TLS for a scenario where a client interacts with a web server. As there are only two affected machines, results in this very simplistic scenario are much more predictable than in the case of cloud database systems.

Shirasuna et al. [12] analyze the performance overhead of TLS in the case of SOAP-based web services. They use a very simple echo service so that the measurement approach cannot be used for the more complex benchmarking of cloud database systems.

Other approaches like [13]–[15] study the TLS performance overhead in combination with protocols not used for cloud database systems, e.g., SIP (Voice over IP) or with IP which is even further down within the ISO/OSI stack.

All these approaches consider only simplistic scenarios, i.e., they compare a single data transfer, e.g., only one RPC call, with and without TLS. In cloud database systems, there are complex interdependencies between different quality of service (QoS) dimensions. Furthermore, changes to the replica-replica communication will also affect application-replica communication. Therefore, a holistic measurement approach for this complex scenario is necessary.

On the other hand, existing approaches for benchmarking of cloud database systems, e.g., [16]–[24], do not consider security mechanisms in their analysis. We, therefore, propose to extend one of the established benchmarking solutions – we chose Yahoo! Cloud Serving Benchmark (YCSB) [16] – to also capture the performance effects of using TLS in various configurations.

III. Measuring the TLS Overhead

In this section, we describe our main contributions, a benchmarking approach for transparent analysis of the performance impact of various TLS configuration options in cloud database systems, as well as a custom TLS socket implementation which offers more fine-grained control over the configuration options chosen. Based on the custom TLS socket (which we implemented in Java) and the existing YCSB [16] research prototype, we developed an implementation of our benchmarking approach. We will make this tool available as open source so that everyone can reproduce the results of our experiments from section IV.

At a first glance, the setup for our measurements of TLS overhead in cloud database systems looks similar to a standard YCSB [16] performance benchmark with encryption enabled (see figure 3). The devil is in the detail, though, since there is an undesirable effect in standard TLS socket implementations: During the handshake phase, the client sends a list of supported cipher suites to the server. If the intersection of the client's cipher suites and the server's supported algorithm combinations contains more than one entry, then a cipher suite is chosen "randomly" (client's favorite choice first which is the first cipher suite in the client's list [7]). Note that this in many TLS implementations is not necessarily the most secure or the fastest algorithm combination.

While this may be a reasonable approach for communication between a web server and a browser, for a benchmarking approach trying to determine the exact performance overhead of TLS it is also necessary to ascertain more fine-grained control over the selection of the cipher suite. The selection of a combination should preferably be based on benchmarking data and a conscious decision instead of being left to chance. This is true for application-replica as well as for replica-replica communication. A way to influence the actual selection of a cipher suite is to limit the list of supported cipher suites provided at one or on both sides of the communication. This can be done by reconfiguring the supported cipher suite list every time a connection is established.

Another strong influence factor is the initial handshake phase itself: If TLS sessions can be reused which results in a connection pooling, this phase is executed just once. If not, it will be added as overhead to every connection establishment between client and server. In Java, this connection pooling for multiple sockets to same communication endpoint is the default option. At the same time, only a limited number of concurrent connections is available so that there is also a tradeoff.

For these reasons, we implemented a custom TLS socket implementation in Java which can be used for both application-replica as well as replica-replica communication. It provides a configurable interface in terms of cipher suites, buffer sizes, etc. and uses the standard Java TLS socket implementation in the background. Hence, the TLS socket is parameterized with the desired combination of cipher suites so that the intersection of sets during the handshake phase contains exactly one element. If the desired combination is not available on the other end, an error is raised instead of establishing a connection via another cipher suite. At the same time, our TLS socket implementation also can easily be extended to offer configurability for degrees of connection pooling.

Beyond our custom TLS socket implementation, there are several additional challenges and influence factors which a measurement approach should consider:

- *Workload and System Configuration:* Depending on the workload and system configuration, the performance impact of TLS may vary. For instance, a quorum system like Cassandra [6] running at consistency level one[14] combined with a read-heavy workload will not be affected by TLS in replica-replica communication if requests do not need to be redirected as a read then does not require any replica-replica communication at all. An update-heavy workload combined with a higher consistency level, on the other hand, will be affected by the full overhead of TLS as every request requires (potentially synchronous) interaction with all replica servers.

[14]The operation commits after reading or writing only one replica; in the case of writes, the remaining replicas are updated asynchronously.

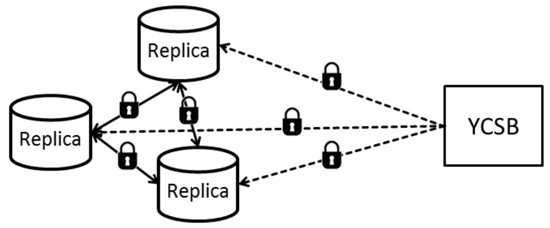

Figure 3. High-level Architecture During Measurements

- *Resource Saturation:* Depending on the degree of resource saturation the observable overhead may vary. For instance, running at low saturation levels we do not expect any influence on the throughput but slightly larger latency values. For very high resource saturation levels, on the other hand, we would expect a reduced throughput value.
- *Comparability:* Determining the overhead of TLS requires running exactly the same benchmark setup with and without TLS. We modified YCSB to do this automatically.
- *Number of Combinations:* There are many encryption and hashing algorithms available with different degrees of vulnerability. The cartesian product of these algorithms results in a very large number of combinations which need to be benchmarked. We are still working on automating these measurements for all combinations.

Combining YCSB extensions with our custom TLS socket implementation, we implemented a benchmarking tool to experimentally determine the performance overhead of TLS in cloud database systems. We also add fine-grained control over security mechanisms in replica-replica communication via our TLS socket implementation[15].

IV. Experiments

We now use our proposed benchmarking approach to study the performance impact of TLS in two different cloud database systems: In Amazon DynamoDB, as an example of a cloud storage service, we could only control the settings for application-replica communication. With Apache Cassandra, as an example of a self-hosted NoSQL system, we could also study how enabling TLS for replica-replica communication affects the client-visible performance.

A. DynamoDB

Amazon DynamoDB is a fully managed cloud storage service with a tabular data structure. Since it does not have a fixed schema, each line may have a different number of columns. For each table a throughput target (i.e., read and write capacity) has to be provisioned in advance, requests beyond the target terminate with an error. As communication middleware, Amazon supports SOAP/WSDL and REST-style web services both on top of HTTP and HTTPS. During the handshake and bulk data transfer phase RSA and RC4 with MD5[16] respectively are used. While this is a very fast combination [13], [25], using this TLS cipher suite is not recommended from a security perspective [26].

[15]As our TLS socket implementation is Java-based, we currently only support this for Java-based replica-replica communication endpoints.

Table II. Experiment setup for DynamoDB.

Setting	Value
Protocol	SSL
Cipher suites	SSL_RSA_WITH_RC4_128_MD5
Workload	Update-heavy not throttled
Handshake renegotiation	False
Packet size	900 B
Number of operations	5,000,000
Initial load	ca. 10 GB
Consistent reads	False

Table III. Experiment Results for DynamoDB.

	No Encryption	RC4
Avg. Throughput in Ops/sec	2,392.4	2,269.5
Std. Dev. in Throughput	323.0	444.2
Avg. Update Latency in ms	9.0	9.2
Std. Dev. Avg. Upd. Latency	1.6	2.1
Min. Update Latency in ms	5.9	6.0
Max. Update Latency in ms	9,461.7	10,133.7
Update Latency 99th Percentile in ms	19.0	19.0
Avg. Read Latency in ms	7.5	7.7
Std. Dev. Avg. Read Latency	1.7	2.1
Min. Read Latency in ms	4.8	4.9
Max. Read Latency in ms	10,341.1	9,933.2
Read Latency 99th Percentile in ms	16.0	15.0

As DynamoDB supports only one cipher suite[16], we compared the performance of DynamoDB with and without this cipher suite. We chose a target throughput of 10,000 units read/write capacity and deployed our benchmarking tool on an m1.large EC2[17] instance. As workload, we decided on an update-heavy workload (50% updates and reads each) which was not throttled to measure the maximum available throughput at the benchmarking machine. Each row was configured to have ten fields at 90 B each so that the total packet size during data transfer was 900 B. See also table II for an overview of the experiment settings.

In our experiments with DynamoDB, which we repeated several times, we could not see any performance impact of TLS since both throughput, as well as read and write latencies showed no statistically significant deviation (see figure 4). Table III gives an overview of our DynamoDB results.

We believe that this can only be explained by Amazon overprovisioning resources so as not to violate their service level agreements (SLA): If TLS is used, resource consumption increases for Amazon without being visible to the client. We, therefore, recommend to use DynamoDB only with TLS activated as it essentially comes for free for the customer.

B. Cassandra

Cassandra is a P2P storage system with a table structure and a flexible schema. The system was originally developed at Facebook[18] as the background system behind their message

[16]TLS cipher suite: SSL_RSA_WITH_RC4_128_MD5
[17]aws.amazon.com/ec2
[18]facebook.com

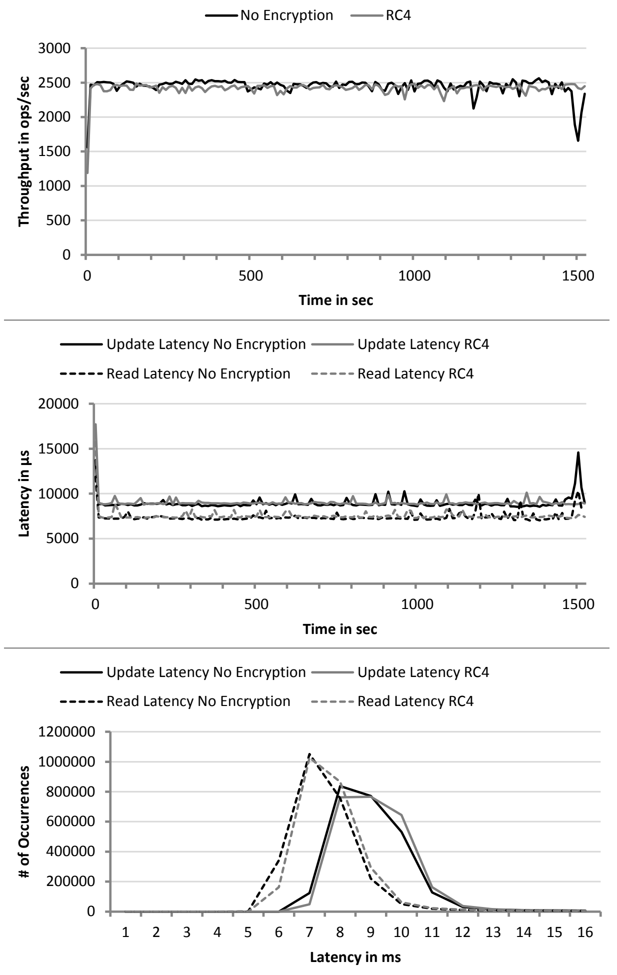

Figure 4. Performance Impact of TLS in DynamoDB

inbox [6] and is now a popular NoSQL system maintained as Apache open source system. For our experiments, we chose Cassandra in version 1.2.9 which uses a custom RPC protocol or Thrift for application-replica communication and a custom RPC protocol for replica-replica communication.

In all our experiments, we deployed Cassandra on a cluster of three m1.large EC2 instances within the same availability zone of the region us-east. We used Thrift for application-replica communication. Where not indicated otherwise, we chose a replication factor and consistency level of one, i.e., there is just one replica for every data item. In this paper, we report results for the following experiments:

- *Experiment AR:* In experiment AR, we measured the performance overhead of using TLS only in application-replica communication.
- *Experiment RR:* In experiment RR, we studied the performance overhead of using TLS in replica-replica communication.
- *Experiment AR-RR:* In experiment AR-RR, we analyzed the performance overhead of using TLS for both application-replica and replica-replica communication.
- *Experiment RR.HL:* In experiment RR.HL, we studied how increasing the load compared to experiment RR affected our results. For this purpose, we used exactly the same setup but deployed a second m1.large EC2 instance with our benchmarking tool to effectively double the load.
- *Experiment RR.BD:* In experiment RR.BD, we studied how our results of experiment RR.HL change when we run the cluster in a Big Data setup. For this purpose, we changed the replication factor to two and consistency level of Cassandra to quorum and increased the data size per item by a factor of 1000.

Table IV. EXPERIMENT SETUP FOR CASSANDRA.

Setting	Value
Protocol	TLS
Cipher suites	TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA
Workload	Update-heavy not throttled
Handshake renegotiation	False
Packet size	1000 B (1000 KB)
Number of operations	3,000,000 (40,000)
Initial load	ca. 28 GB (ca. 150 GB)
Consistency level	ONE (QUORUM)
Replication Factor	1 (2)

In each experiment, we compared the performance without TLS to the performance of TLS with AES 128 and 256[19] to also study whether the key length has a visible influence on performance. We chose these cipher suites as they used to be a NIST recommendation until March 2013 and are still widely used in practice. For all experiments, we again ran an unthrottled, update-heavy workload (50% updates and reads each) against an initial data set of about 30 GB. We used connection pooling, i.e., the TLS handshake happened in each experiment only once per Cassandra node and client (3 handshakes for experiments AR and RR, 6 handshakes for experiment AR-RR; all connections were reset before and after the switch between no TLS, AES 128 and AES 256). Table IV gives an overview of the setup for all experiments; values in brackets describe the differing setup in experiment RR.BD.

1) Experiment AR: Our results show a significant performance impact of enabling TLS in application-replica communication (see figure 5 and table V), while the key length in AES seems to have no influence. Interestingly, update latencies seems unaffected (difference: $< 1\,ms$) whereas read latencies are higher (difference: $1\,ms$) and average throughput of AES 256 is about 18.9% (368 Ops/sec difference) lower compared to application-replica communication without TLS.

When repeating this experiment, "anomalies", where secured communication suddenly outperforms application-replica communication without TLS (e.g., the throughput chart in figure 5 after about 1000 seconds), kept reoccurring. Since there was no clear tendency whether AES 128 or 256 or unsecure communication showed this anomaly more frequently, we believe that this can only be explained by the general performance variance of public cloud resources [27]–[29]. In the results which we show here, the anomaly obviously affects the average values of table V.

[19]Cipher suites: TLS_DHE_RSA_WITH_AES_128_CBC_SHA and TLS_DHE_RSA_WITH_AES_256_CBC_SHA

Table V. Experiment Results for Experiment AR

	No Encryption	AES 128	AES 256
Avg. Throughput in Ops/sec	1945.3	1760.4	1576.9
Std. Dev. in Throughput	264.6	493.4	416.5
Avg. Upd. Lat. in ms	3.3	3.5	3.9
Std. Dev. Avg. Upd. Lat.	0.7	1.3	1.1
Min. Upd. Lat. in ms	0.4	0.5	0.5
Max. Upd. Lat. in ms	686.9	2,986.5	1,411.3
Upd. Lat. 99th Perc. in ms	36.0	35.0	39.0
Avg. Read Lat. in ms	26.7	30.0	33.8
Std. Dev. Avg. Read Lat.	4.6	11.3	12.1
Min. Read Lat. in ms	0.6	0.8	0.8
Max. Read Lat. in ms	1,391.7	3,201.6	1,932.7
Read Lat. 99th Perc. in ms	255.0	312.0	373.0

Another observation which we can draw from this, is that the reliability of the performance decreases when using TLS as there is a much higher variance with TLS than without (see the standard deviation values in table V).

2) Experiment RR: Since we have not been able to see any influence of the key length of AES, we ran this and all following experiments only with AES 256[20] instead of both AES 128 and 256.

In experiment RR, where all replica-replica communication used TLS, we could not see a statistically significant impact of using TLS-secured replica-replica communication on the performance of our Cassandra cluster: Neither throughput nor update and read latencies show any effect (see figure 6). We believe that the impact of TLS on the performance of the replica-replica communication itself is, in this experiment, too low to be visible to the end client. As we will see in section IV-B4, this may look different at a higher resource saturation level of the cluster.

One might argue that with a replication factor of one, there is no replica-replica communication. This is not true as Cassandra is obviously not designed for geo-distribution: Each Thrift request arriving at node A requiring data from node B will be processed by node A which requests the data from node B and then forwards B's response to the client again via Thrift. I.e., there is no separation of data and control flow as the data from B will be funneled through A instead of sending a redirect for node B to the client. Therefore, as long as not every request directly reaches the correct node on the ring (e.g., with three nodes, a replication factor of three and a consistency level of one), there is replica-replica communication even with a replication factor of one.

3) Experiment AR-RR: In experiment AR-RR, we activated TLS for both application-replica and replica-replica communication. After having seen the results of experiments AR and RR, the results were not surprising: There is an overhead which is visible in both throughput and latency values. Also, we could again observe the anomaly already discussed in section IV-B1. We expected this behavior, since adding no overhead (experiment RR) and some overhead plus an anomaly (experiment AR) is likely to show the exact combination – an overhead with an anomaly. Again, we could reproduce this anomaly in either direction.

[20]Cipher suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA

Table VI. Experiment Results for Experiment AR-RR

	No Encryption	AES 256
Avg. Throughput in Ops/sec	1945.3	1769.8
Std. Dev. in Throughput	264.6	390.0
Avg. Upd. Lat. in ms	3.3	4.5
Std. Dev. Avg. Upd. Lat.	0.7	1.4
Min. Upd. Lat. in ms	0.4	0.5
Max. Upd. Lat. in ms	686.9	540.1
Upd. Lat. 99th Perc. in ms	36.0	43.0
Avg. Read Lat. in ms	26.7	29.1
Std. Dev. Avg. Read Lat.	4.6	8.9
Min. Read Lat. in ms	0.6	0.8
Max. Read Lat. in ms	1,391.7	1,178.2
Read Lat. 99th Perc. in ms	255.0	261.0

Table VII. Experiment Results for Experiment RR.HL

	No Encryption		AES 256	
	YCSB1	YCSB2	YCSB1	YCSB2
Avg. Throughput in Ops/sec	1898.4	1903.8	1587.5	1586.5
Std. Dev. in Throughput	366.4	341.4	250.1	249.0
Avg. Upd. Lat. in ms	2.0	2.0	3.1	3.3
Std. Dev. Avg. Upd. Lat.	0.5	0.5	0.5	0.5
Min. Upd. Lat. in ms	0.5	0.5	0.5	0.5
Max. Upd. Lat. in ms	574.7	612.8	1,017.3	1,018.0
Upd. Lat. 99th Perc. in ms	8.0	8.0	15.0	16.0
Avg. Read Lat. in ms	18.8	18.7	21.5	21.5
Std. Dev. Avg. Read Lat.	8.4	7.0	8.5	6.7
Min. Read Lat. in ms	0.7	0.7	0.7	0.7
Max. Read Lat. in ms	2,056.9	2,012.5	3,644.5	3,598.9
Read Lat. 99th Perc. in ms	314.0	313.0	313.0	314.0

Figure 7 and table VI show the results of experiment AR-RR. One can clearly see that using TLS again considerably affects the variance values.

4) Experiment RR.HL: After not being able to observe a client-visible performance impact when using TLS in replica-replica communication, we doubled the load on our Cassandra cluster by running a second machine with our benchmarking client in parallel.

With the increased number of parallel requests to the cluster, we managed to increase the CPU load of the cluster's machines from around 40% to 60%. Obviously, the resource saturation of the storage cluster has a large influence on whether secure replica-replica communication has a client-visible performance impact: Throughput decreased by about 300 Ops/sec and the average latencies of updates and reads increased by about 1 ms and 3 ms respectively. It is, hence, safe to say that a higher degree of resource saturation increases the severity of TLS performance impacts. We expect even higher impacts for higher resource saturation levels so that the decision on using or not using TLS should also be based on the expected utilization level of the cluster.

For an overview of the results of this experiment, see table VII and figure 8.

5) Experiment RR.BD: In our Big Data experiment, we drastically increased the field size, so that every row stored 10 fields of 512,000 characters each as it might be the case if Cassandra was used, for instance, to store larger texts for data mining purposes. We also increased the replication level and

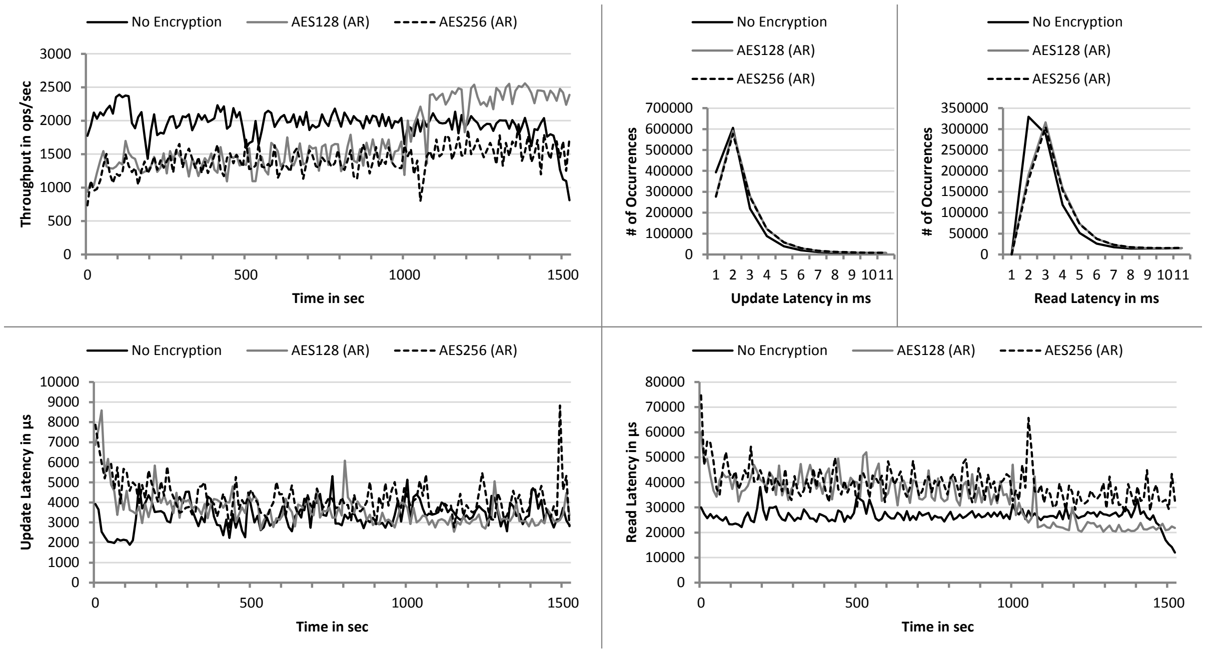

Figure 5. Performance Impact of TLS in Application-Replica Communication in Experiment AR

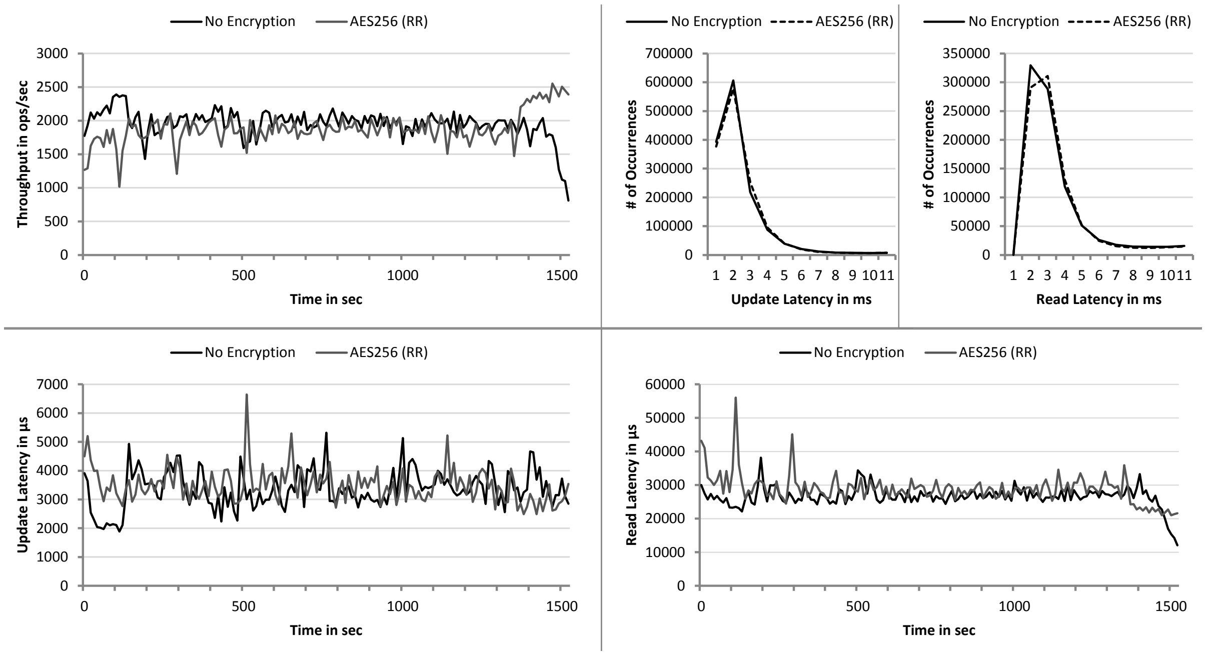

Figure 6. Performance Impact of TLS in Replica-Replica Communication in Experiment RR

quorum size both to two, so as to enforce more replica-replica communication. Table IV shows in brackets the changes we made to the original experiment setup.

In this experiment, we observed a rather curious behavior (see table VIII and figure 9) where activating TLS for replica-replica communication considerably increases throughput from about 17.5 to 22.25 Ops/sec whereas latencies are much worse. Also, the reliability decreases as we could see a jump in variance. We repeated this experiment three times, starting at different times during the day, and each time saw the same effect of increased throughput. We believe that this may be due to an optimal fit of TCP packet sizes and the selected data sizes. Still, we need to study this more also for other payload sizes and characteristics to determine the root cause behind this effect.

6) Further Experiments: The experiments reported within this paper were repeated several times, each time yielding comparable results. Beyond these experiments, we also ran additional benchmarks where we varied the replication level and quorum sizes (i.e., the consistency level of Cassandra) to increase communication effort. Furthermore, we used different workload types (workloads A, B and C of YCSB [16])

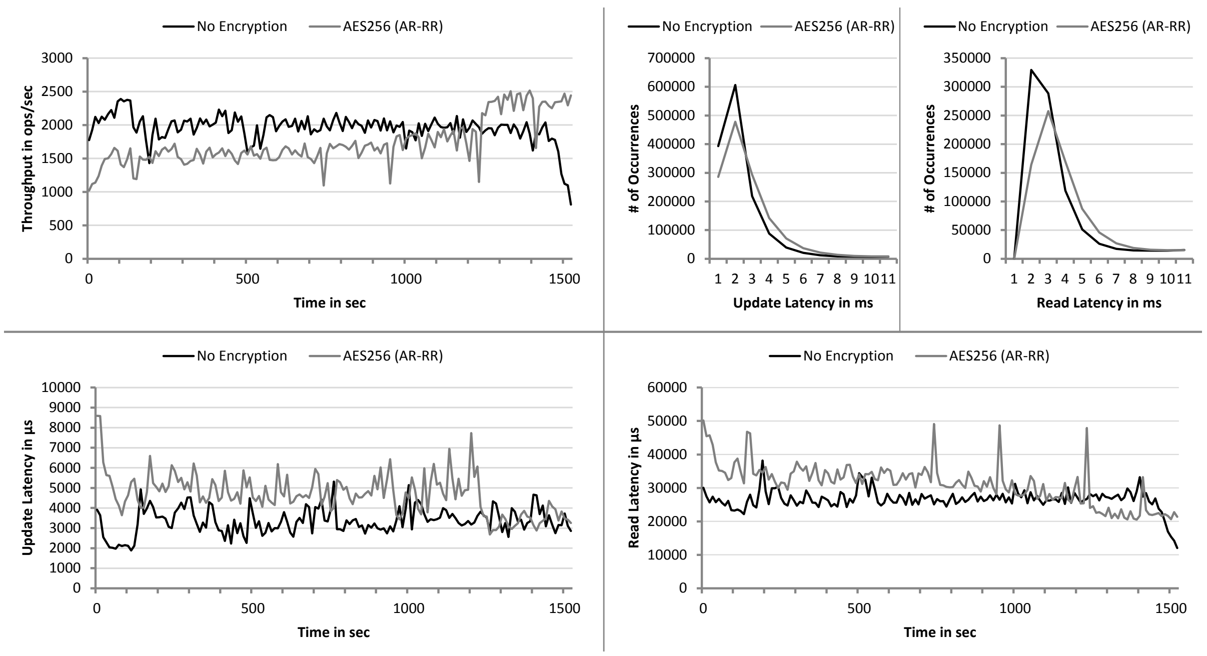

Figure 7. Performance Impact of TLS in both Application-Replica and Replica-Replica Communication in Experiment AR-RR

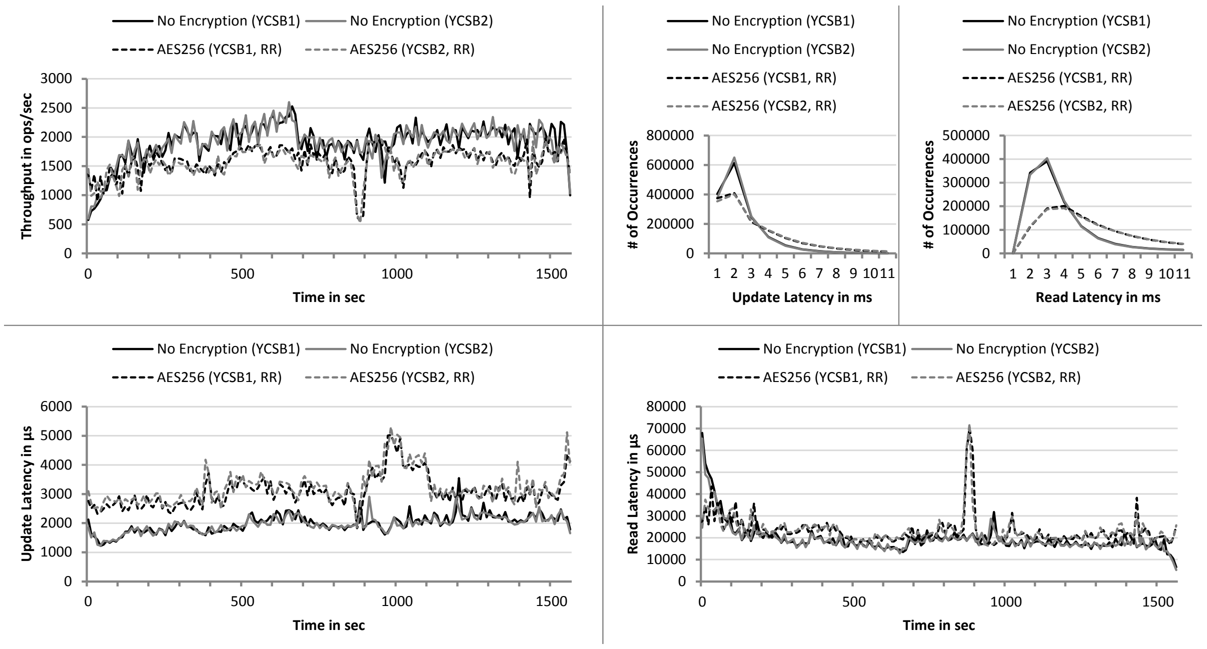

Figure 8. Performance Impact of TLS in Replica-Replica Communication in Experiment RR.HL

with a varying number of fields. None of these additional experiments showed significant deviations from the behavior described above; results were comparable to what we would have expected after having seen the results of experiments AR, RR, AR-RR, RR.HL, and RR.BD.

Finally, we reran a subset of our experiments on cloud resources of ProfitBricks[21]. As we have observed, the ProfitBricks cloud offers a larger and more stable network bandwidth which could also be seen in our results: While being similar to our Amazon Web Services results, the differences with and without TLS were more severe, throughput and latency showed less variance, no unexplainable anomalies occurred, and measurement results generally showed better performance.

[21]profitbricks.com

V. Discussion and Conclusion

In this paper, we presented an approach for benchmarking the performance impact of using TLS in cloud database systems. We developed a custom TLS socket implementation as well as an extension of the benchmarking tool YCSB which together allow us to conduct fine-grained experiments with

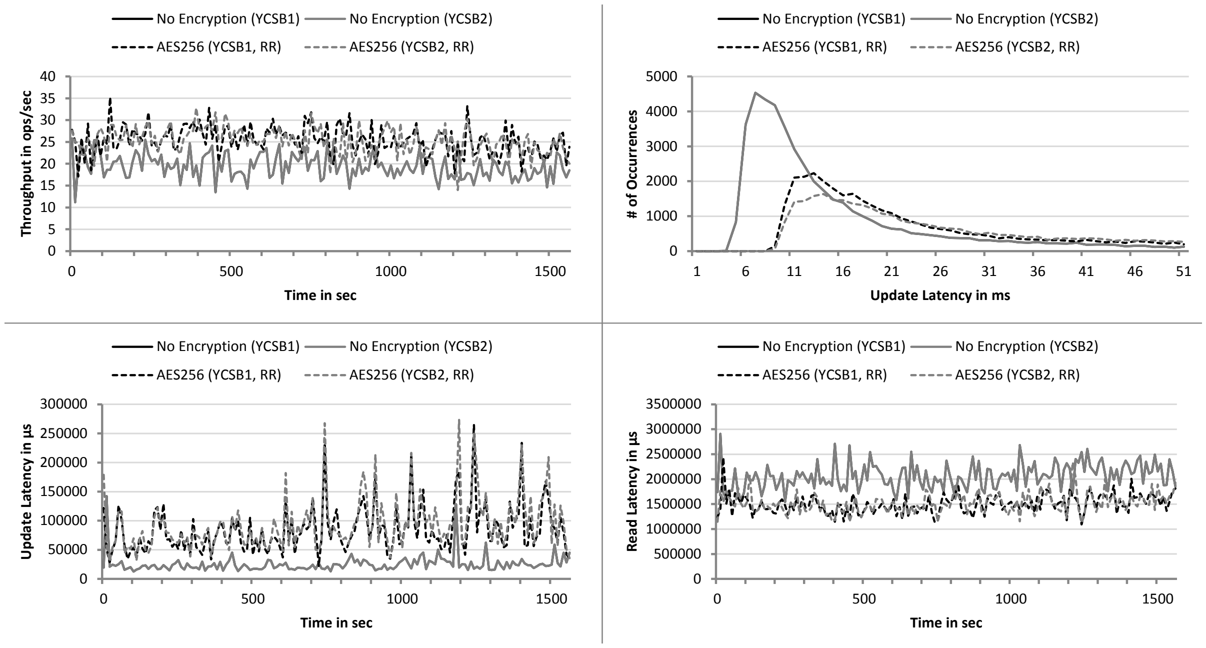

Figure 9. Performance Impact of TLS in Replica-Replica Communication in Experiment RR.BD

Table VIII. Experiment Results for Experiment RR.BD

	No Encryption		AES 256	
	YCSB1	**YCSB2**	**YCSB1**	**YCSB2**
Avg. Throughput in Ops/sec	17.49	17.49	22.28	22.24
Std. Dev. in Throughput	2.9365	2.9365	4.1196	3.8467
Avg. Upd. Lat. in ms	23.1	25.5	75.3	84.0
Std. Dev. Avg. Upd. Lat.	12.3	12.3	38.7	45.6
Min. Upd. Lat. in ms	3.8	4.0	8.3	8.1
Max. Upd. Lat. in ms	3,625.6	2,485.1	2,178.7	2,186.5
Upd. Lat. 99th Perc. in ms	262	259	607	643
Avg. Read Lat. in ms	2,225.7	2,248.5	1,678.6	1,680.8
Std. Dev. Avg. Read Lat.	361.6	361.6	310.3	321.7
Min. Read Lat. in ms	43.2	8.0	45.3	42.4
Max. Read Lat. in ms	13,660.9	22,061.3	9,946.9	11,020.7

different TLS configurations for application-replica as well as replica-replica communication.

Based on our tool, we conducted several experiments with two common cloud database systems: Amazon DynamoDB and an Apache Cassandra cluster running in the cloud. As the experiments show, the use of TLS in application-replica communication has a measurable impact on Cassandra's performance whereas the impact of TLS for replica-replica communication was statistically significant only in cases with higher resource saturation. A highly interesting, though yet unexplained effect, appeared in the setup with drastically increased field size: Here, the activation of TLS for replica-replica communication led to a significantly *increased* throughput, albeit at the cost of significantly worse latencies. Amazon's DynamoDB, in turn, showed no significant performance impact resulting from the activation of TLS.

These results lead to some preliminary practical implications as well as to a couple of open research questions. First, there is a proven and significant perfomance impact of TLS being activated in practically relevant cloud database systems. This impact can be measured experimentally and should be accounted for in concrete decisions having to balance security against performance (or, respectively, costs). Even if the impact strongly depends on the concrete setup and workload profile, it can easily be determined for specific use cases based on our methodology and tools.

As the impact of TLS on Cassandra's replica-replica communication could only be observed in high-load scenarios, one might conclude that this is less relevant due to the general recommendation not to operate cloud databases at their performance limits. But depending on the use case, operation at the performance limit might very well make sense (e.g., from a cost perspective). At least for these cases, the activation of TLS for replica-replica communication should be based on deliberations considering figures as provided by our approach and tools. For all other cases, the activation of TLS for replica-replica communication should at least be considered as an additional factor in determining the available overall performance and thereby possibly calling for increased resources to *prevent* performance limits from being reached.

So far we did not run any experiments with renegotiation of TLS sessions because we think that a cloud developer should avoid repeatedly closing and opening a cloud database's connection. Frequent session renegotiation might, however, be necessary in certain use cases. As mentioned before, we implemented this feature into our tools for testing purposes and it might be interesting to measure the influence of repeated session renegotiation in future experiments.

The concrete impact of using TLS-based communication will, however, in all likelihood vary across different cloud database systems. Due to the highly different implementations of application-replica as well as replica-replica communication across different systems, our results will hardly be transferable to other systems than those considered herein. Comparable experiments should therefore be conducted for other cloud database systems like HBase or Voldemort in order to allow for a more comprehensive view on the performance impact of TLS in cloud database systems.

Last but not least, this broadening of the experimental base should not only cover different cloud database systems but different cloud providers as well. As we have seen in our preliminary experiments in this regard, the specific givens of different providers can have significant influence at least on the measured variance and the occurence of anomalies. This might in turn constitute another relevant factor in deciding on concrete system architectures involving cloud database systems. Again, our methodology and tools might prove valuable here, too.

Acknowledgment

The authors would like to thank Amazon Web Services and ProfitBricks who provided research grants for our experiments.

References

[1] D. Bermbach and J. Kuhlenkamp, "Consistency in distributed storage systems: An overview of models, metrics and measurement approaches," in *Proc. of NETYS 2013*, 2013.

[2] E. A. Brewer, "Towards robust distributed systems," in *PODC 2000 Keynote*, 2000.

[3] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, feb. 2012.

[4] W. Vogels, "Eventually consistent," *Queue*, vol. 6, pp. 14–19, October 2008.

[5] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] T. Dierks and E. Rescorla, "Rfc5246: The transport layer security (tls) protocol version 1.2," 2008. [Online]. Available: http://tools.ietf.org/rfc/rfc5246.txt

[8] G. Apostolopoulos, V. Peris, and D. Saha, "Transport layer security: how much does it really cost?" in *Proc. of INFOCOM 1999*, 1999, pp. 717–725.

[9] K. Kant, R. Iyer, and P. Mohapatra, "Architectural impact of secure socket layer on internet servers," in *Proc. of Computer Design 2000*, 2000, pp. 7–14.

[10] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, "Anatomy and performance of ssl processing," in *Proc. of ISPASS 2005*, march 2005, pp. 197–206.

[11] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of tls web servers," *ACM Trans. Comput. Syst.*, vol. 24, no. 1, pp. 39–69, Feb. 2006.

[12] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon, "Performance comparison of security mechanisms for grid services," in *Proc. of GRID 2004*, 2004, pp. 360–364.

[13] S. Rapuano and E. Zimeo, "Measurement of performance impact of ssl on ip data transmissions," *Measurement*, vol. 41, no. 5, pp. 481–490, 2008.

[14] C. Shen, E. Nahum, H. Schulzrinne, and C. Wright, "The impact of tls on sip server performance," in *Principles, Systems and Applications of IP Telecommunications*, 2010, pp. 59–70.

[15] M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko, "Comparison of performance of web services, ws-security, rmi, and rmiŰssl," *Journal of Systems and Software*, vol. 79, no. 5, pp. 689–700, 2006.

[16] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proc. of SOCC 2010*, 2010, pp. 143–154.

[17] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi, "Ycsb++: benchmarking and performance debugging advanced features in scalable table stores," in *Proc. of SOCC 2011*, 2011, pp. 1–14.

[18] L. Zhao, A. Liu, and J. Keung, "Evaluating cloud platform architecture with the care framework," in *Proc. of Asia Pacific Software Engineering Conference 2010*, 2010, pp. 60–69.

[19] D. Bermbach, S. Sakr, and L. Zhao, "Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services," in *Proc. of TPCTC 2013*, 2013.

[20] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior," in *Proc. of 6th Workshop on Middleware for Service Oriented Computing*, 2011, pp. 1–6.

[21] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: towards a benchmark for the cloud," in *Proc. of Int. Workshop on Testing Database Systems 2009*, 2009, pp. 1–6.

[22] E. Anderson, X. Li, M. Shah, J. Tucek, and J. Wylie, "What consistency does your key-value store actually provide," in *Proc. of HotDep 2010*, 2010, pp. 1–16.

[23] M. Klems, D. Bermbach, and R. Weinert, "A runtime quality measurement framework for cloud database service systems," in *Proc. of QUATIC 2012*, 2012, pp. 38–46.

[24] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1724–1735, Aug. 2012.

[25] D. Menasce, "Security performance," *Internet Computing, IEEE*, vol. 7, no. 3, pp. 84–87, may-june 2003.

[26] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. Schuldt, "On the security of rc4 in tls and wpa," in *USENIX Security Symposium*, 2013.

[27] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[28] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. of ACM SIGCOMM 2011*, 2011, pp. 242–253.

[29] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? performance benchmarking for infrastructure-as-a-service offerings," in *Proc. of CLOUD 2011*, 2011, pp. 484–491.