BALANCING PRIVACY, PRECISION AND PERFORMANCE IN DISTRIBUTED SYSTEMS

by

Marian K. Zaki

B.Sc., Ain Shams University, Cairo, Egypt 2002

M.Sc., Ain Shams University, Cairo, Egypt 2006

Submitted to the Graduate Faculty of

the Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Marian K. Zaki

It was defended on

November 8th, 2019

and approved by

Dr. Adam J. Lee, Associate Professor, Department of Computer Science

Dr. Panos K. Chrysanthis, Professor, Department of Computer Science

Dr. Daniel Mossè, Professor, Department of Computer Science

Dr. Vladimir I. Zadorozhny, Professor, Department of Informatics and Networked Systems

Dissertation Director: Dr. Adam J. Lee, Associate Professor, Department of Computer Science

BALANCING PRIVACY, PRECISION AND PERFORMANCE IN DISTRIBUTED SYSTEMS

Marian K. Zaki, PhD

University of Pittsburgh, 2019

Privacy, Precision, and Performance (3Ps) are three fundamental design objectives in distributed systems. However, these properties tend to compete with one another and are not considered absolute properties or functions. They must be defined and justified in terms of a system, its resources, stakeholder concerns, and the security threat model.

To date, distributed systems research has only considered the trade-offs of balancing privacy, precision, and performance in a pairwise fashion. However, this dissertation formally explores the space of trade-offs among all 3Ps by examining three representative classes of distributed systems, namely Wireless Sensor Networks (WSNs), cloud systems, and Data Stream Management Systems (DSMSs). These representative systems support large part of the modern and mission-critical distributed systems.

WSNs are real-time systems characterized by unreliable network interconnections and highly constrained computational and power resources. The dissertation proposes a privacy-preserving in-network aggregation protocol for WSNs demonstrating that the 3Ps could be navigated by adopting the appropriate algorithms and cryptographic techniques that are not prohibitively expensive.

Next, the dissertation highlights the privacy and precision issues that arise in cloud databases due to the eventual consistency models of the cloud. To address these issues, consistency enforcement techniques across cloud servers are proposed and the trade-offs between 3Ps are discussed to help guide cloud database users on how to balance these properties.

Lastly, the 3Ps properties are examined in DSMSs which are characterized by high volumes of unbounded input data streams and strict real-time processing constraints. Within this system, the 3Ps are balanced through a proposed simple and efficient technique that applies access control policies over shared operator networks to achieve privacy and precision without sacrificing the systems performance.

Despite that in this dissertation, it was shown that, with the right set of protocols and algorithms, the desirable 3P properties can co-exist in a balanced way in well-established distributed systems, this dissertation is promoting the use of the new *3Ps-by-design* concept. This concept is meant to encourage distributed systems designers to proactively consider the interplay among the 3Ps from the initial stages of the systems design lifecycle rather than identifying them as add-on properties to systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

PREFACE

To every thing there is a season, and a time to every purpose under the heaven.

Ecclesiastes 3:1

There are many people who I owe gratitude for more than what can be put in words. To start, I would like to thank my advisor Dr. Adam J. Lee for supporting me for many years during all my ups and downs and for showing me how to be a better researcher and teacher. I owe Adam for most of what I have learned during my Ph.D. years and he is and will always be a role model to me. I also extend my thanks to my co-advisor Dr. Panos K. Chrysanthis for his support and constructive feedback on most of the work presented in this dissertation. I would like to thank Dr. Daniel Mossè who encouraged me to start my research from the very first semester as a graduate student and for nominating me for the teaching mentor award and position I earned.

I am eternally thankful to my husband Ragae, he believed in me more than I did in myself. Without his encouragement and patience in this long journey that we took together, this dissertation would have not been achieved. I am also grateful and thankful for my family, my mom and dad, who are in heaven now. They always taught me to work hard and do the best I can to achieve my goals. Despite they are unable to witness this achievement, I know that they would not be any prouder to see their youngest daughter graduate from the same University they graduated from. I am thankful for my sister Manal, her husband and daughters for their continuous encouragement. Life became a lot easier with them supporting my family from the first day we moved to the states.

Upove all, I give thanks to the Lord for granting me with His grace and love more than what I asked for.

Dedicated to my daughters Doris and Emmy. They are the reason for my perseverance to finish what I started. To them I say, “Never give up easily on your dreams and goals no matter how long the journey takes, trust the Lord and He will guide you always”.

1.0 INTRODUCTION

In today's networked world, distributed systems are becoming the norm and are widely used throughout the globe. They are used in a variety of applications to distribute work, speed up data processing, or simply collect and store data. In the literature, researchers have used diverse definitions to outline what a distributed system is. Coulouris et al. [48], have defined a distributed system as, "a system where the hardware and software components have been installed in geographically dispersed computers that coordinate and collaborate their actions by passing messages between them." Tanenbaum and Van Steen [137] have defined a distributed system as, "a collection of systems that appears to the users as a single system." These definitions show that distributed systems are more than just communication networks. Their popularity was gained from their ability to connect users and share different resources in a transparent, open, and scalable way to provide users with a single and integrated coherent network.

Many modern distributed systems are required to scale in terms of their support for processes, resources, and users and are often required to operate across the Internet and across different administrative domains. As connectivity becomes more pervasive, many distributed systems demonstrate the need to interface to services and other distributed systems that were not there at design time. This continuous growth brought forth a plethora of design challenges that were never encountered in centralized or stand-alone systems.

It is interesting to look at the trend of how distributed systems challenges have been tackled in the past. In complete isolation, systems designers, developers and even researchers would introduce solutions to one challenge just to create more problems and issues in other aspects of the distributed system. For example, some of the most commonly known challenges in the design of distributed systems are reliability and fault-tolerance [28]. Ideally,

distributed systems are drastically more fault tolerant and more powerful than centralized or stand-alone computer systems, but achieving fault-tolerance is costly in several ways. The cost factor is introduced when distributed systems' designers rely on the redundancy of the distributed system components, such as the hardware components or the data duplication, to tolerate failures.

On the other hand, the heavy use of redundancy of some components in distributed systems induced data imprecision problems. Many distributed systems rely on weak or eventually consistent replication of data to improve systems performance, rather than enforcing strict consistency over multiple copies, which may cause imprecise retrieval of data [143, 118]. For example, consider medical records outsourced to cloud servers with a weak consistency model of data replicated over the different cloud servers, medical professionals could end up querying copies of the data that may be arbitrarily out of date.

Another example of the cascading effect of challenges in distributed systems can be demonstrated as these systems continue to move out into open environments as part of their scalability requirements creating a vast increase in data processing and the distribution of processing across multiple sites. The more input channels and links are created, the more security and data privacy concerns start to rise. Protecting communications, managing authentication and access control to ensure the privacy of the users become fundamental concerns. Unfortunately, these scalability requirements lead to a number of challenges in which the security and privacy needs to be traded off against loss of performance.

The pace at which distributed systems continue to advance was, is, and continues to be overwhelming despite all the design and implementation challenges. The benefits of distributed systems applications are many, making it worthwhile to continue pursuing their development. The real challenge now in the design of distributed systems lies in exploring the trade-offs between some of the design goals and providing an understanding of how design choices can be made to balance these trade-offs.

1.1 PRIVACY, PRECISION AND PERFORMANCE (THE 3PS)

This dissertation focuses on three orthogonal desirable properties that need to co-exist in distributed systems, namely, privacy, precision and performance. One would like a system that enforces *privacy* by not exposing data to unauthorized entities. At the same time, one would prefer a system that ensures the *precision* of the data, i.e., protects the data from loss, provides accurate aggregates and correct query results and so on. And of course one does not want to sacrifice *performance*. There is a very wide spectrum of these properties. Unfortunately, scoring "high" marks in all of the 3P properties in any distributed system is a non-trivial task, if even possible.

Clearly, there is a mutual trade-off between all 3P properties. The first trade-off between privacy and performance arises from the fact that implementing high security systems involves complex cryptographic computations and expensive communications between the elements of the distributed system, which negatively impacts performance. Yet, most lightweight distributed systems have very simplified designs with none or only basic security measures implemented (such as simple user authentication and login functionalities). The second trade-off between data precision and performance can be found in distributed systems that tackle performance problems through replication, by which components are copied and placed close to where they are needed. The problem arises as soon as the read-to-update ratios decrease, replication may actually lead to availability and performance issues of the system if consistency is needed [141]. Accordingly, most distributed systems switch to weak consistency models which in turn can cause data imprecision. This follows from the CAP theorem that states that simple consistency and availability cannot be combined in the presence of network partitions [63]. Finally, a trade-off between privacy and precision exists as well. In modern distributed systems that are hosted by multiple administrative organizations to build collaborative systems, data privacy concerns can often prevent organizations from sharing data for data mining or query processing purposes which affects both the data precision and the utility of the data in general [94].

It should be clear by now that developing a distributed system that balances the conflicting 3P properties can be a formidable task. There are so many issues to consider at the

same time that it seems that only complexity can be the result. While all of 3P properties have been studied individually or pairwise extensively, no previous work has considered the interplay among all three of these attributes. This raises the question that is being addressed in this dissertation:

"Is it possible to develop distributed systems that can balance among the three conflicting attributes, Privacy, Precision, and Performance (3Ps)?"

This question arises in a number of applications and distributed systems, yet each system has its own interpretation of the 3Ps. This means that the 3Ps are not "absolute" properties. They need to be defined and justified based on each distributed system assumptions, constraints and needs. The concrete definition of the 3Ps has to be set correctly for each system to be able to device the right techniques to achieve the desired balance among the 3Ps. Therefore, this dissertation examines three representative classes of distributed systems where the 3P attributes need to be well-defined and the interplay between them needs to be understood and evaluated. Since scoring "high" marks on all three attributes is an unrealistic requirement, this dissertation focuses on finding an acceptable middle ground to provide guarantees on achieving a good balance among the 3Ps.

1.2 THESIS STATEMENT AND CONTRIBUTIONS

This dissertation contributes to the distributed systems field by presenting, for the first time, a cohesive study of the interplay between the conflicting 3P goals in three different classes of distributed systems. It provides novel integrated solutions to applications that are part of the representative broader classes of distributed systems. These solutions empower the design of these applications and make them more trustworthy in terms of users privacy and data precision without penalizing the performance.

The goal of this dissertation is to provide sufficient evidence to satisfy the following statement:

None of the 3P properties should be abdicated to preserve the other two. Balancing the three properties in distributed systems is essential and achievable when the right set of techniques and algorithms are used in harmony with the systems' constraints and requirements.

Successfully justifying the above claim will effectively bridge the gap between the theoretical design principals that state the need for the 3Ps and the practical approaches of how to achieve them all in a balanced way. To accomplish this goal, this dissertation studies the trade-offs of the 3Ps in three different classes of distributed systems. Figure 1 is an illustrative diagram of the three distributed systems that are presented in depth in the following chapters of the dissertation, namely, Wireless Sensor Networks (WSNs), cloud distributed systems, and Data Stream Management Systems (DSMSs). While the applications presented for these distributed systems display certain differences in their constraints and requirements, they all demonstrate one commonality, which is their need to achieve a balance among the 3Ps.

These systems were chosen by this dissertation as representative of some important classes of applications in distributed systems. Each system operates within a given set of constraints and design challenges that makes balancing all 3P properties an interesting problem to investigate. The following is a brief description of each system and how it represents a broader class of applications in distributed systems:

- **Wireless Sensor Networks (WSNs)**: WSNs represent a broad class of distributed systems that operate within very tight constrained resources such as energy, bandwidth and computational power. Given the potential size of the data generated by sensors that could possibly have privacy concerns, a balance between the 3Ps will be required to enable the in-network processing and aggregation of the data in a secure fault tolerant way while considering the high resource constraints. In particular, this dissertation studies the balancing of 3Ps in an example application of WSNS: Collaborative WSNs(CWSNs).

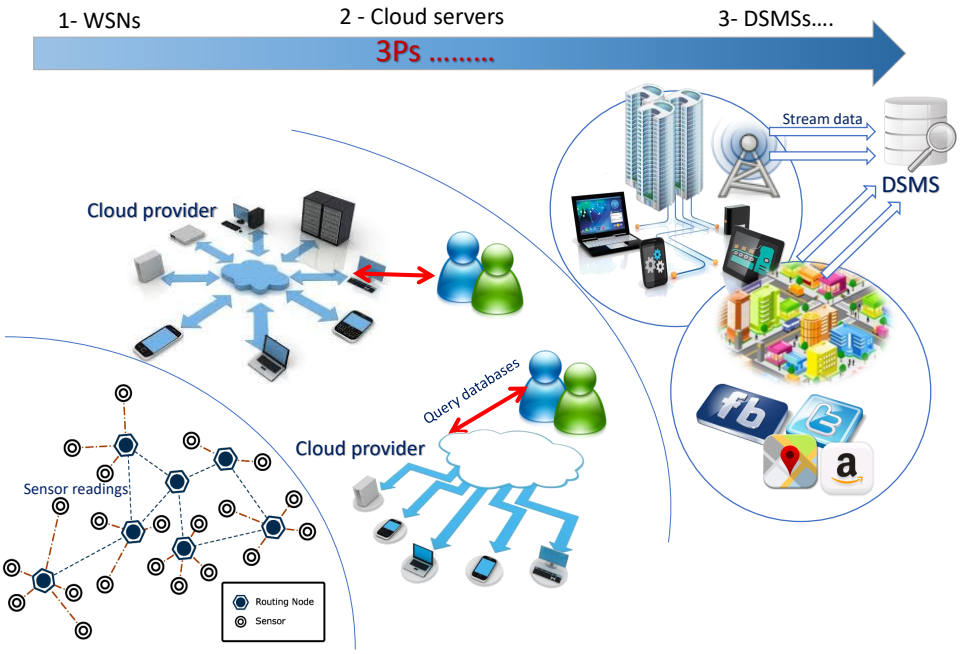


Figure 1: Illustration of the three representative distributed systems

- **Cloud Systems**: As systems continue to move and migrate processing and data on the cloud, a new class of distributed systems has emerged. This class of distributed systems applications provide higher levels of scalability, availability and elasticity of both the data and computing services than any other distributed system. Despite the popularity of cloud services and their wide adoption by enterprises and governments, cloud providers still lack services that guarantee a balance among the 3Ps. This dissertation presents a novel integrated solution to achieve a balance across all 3Ps requirements in one example commonly used application in the cloud: transactional cloud databases.

- **Data Stream Management Systems (DSMSs)**: DSMSs represent a new class of distributed systems that require real-time processing of very high input rates of datastreams. DSMSs need to operate in memory to meet real-time requirements as opposed to cloud databases. This dissertation presents a simple and efficient solution for balancing all 3Ps properties to deliver applications that can efficiently and safely use DSMSs to query datastreams.

The research presented in this dissertation makes a number of contributions in the design of these classes of distributed systems that demonstrate distinctive constraints and requirements. In examining whether the 3Ps could be faithfully balanced in representative applications in each of the three classes of distributed systems, the following contributions are being made:

1. A protocol design for privacy-preserving and fault-tolerant in-network data aggregation for deployments of CWSNs is presented. Privacy of individual sensor readings from malicious or compromised sensor nodes is achieved through end-to-end encryption of individual sensor readings using homomorphic cryptographic systems [33]. Fault-tolerance protocols are used to recover from communication link failures and ensure high data precision of the final aggregate results in the network. Performance is evaluated using simulations to study the effectiveness of the proposed protocol. Simulation results showed minimal induced overheads on the overall energy and bandwidth consumption of the network [80].
2. The new concept of *safe transactions* is introduced in transactional databases executing in the cloud. Safe transactions ensure the consistency of both data and access control policies during the lifetime of a transaction. Safe transactions ensure both privacy and precision of data. Different levels of policy consistency constraints and their enforcement approaches are proposed [81, 82]. A Two-Phase Validation Commit (2PVC) protocol is proposed to ensure that transactions are safe by checking policy, credential, and data consistency during transaction execution. The levels of consistency constraints range from less stringent (high performance) to high stringent (lower performance). These levels show interesting tradeoffs between the 3P requirements and provide cloud application designers and developers the capability of tuning to the consistency levels they want based on their application performance needs.
3. An efficient and simple solution to the problem of enforcing access control in DSMSs is presented. The solution allows access control policies to be applied over multi-query execution plans without the need to modify these plans at runtime whenever access control changes are encountered. This ensures that both data privacy and the precision of query results are both maintained at all times regardless of the current state of the

access control policies or the user credentials. By saving the system from continuously fluctuating between different execution plans for the queries, due to the potential changes in access control policies, the runtime system costs are saved and better performance is achieved. Simulations show the the proposed solution adds negligible overheads given the substantial runtime savings offered.

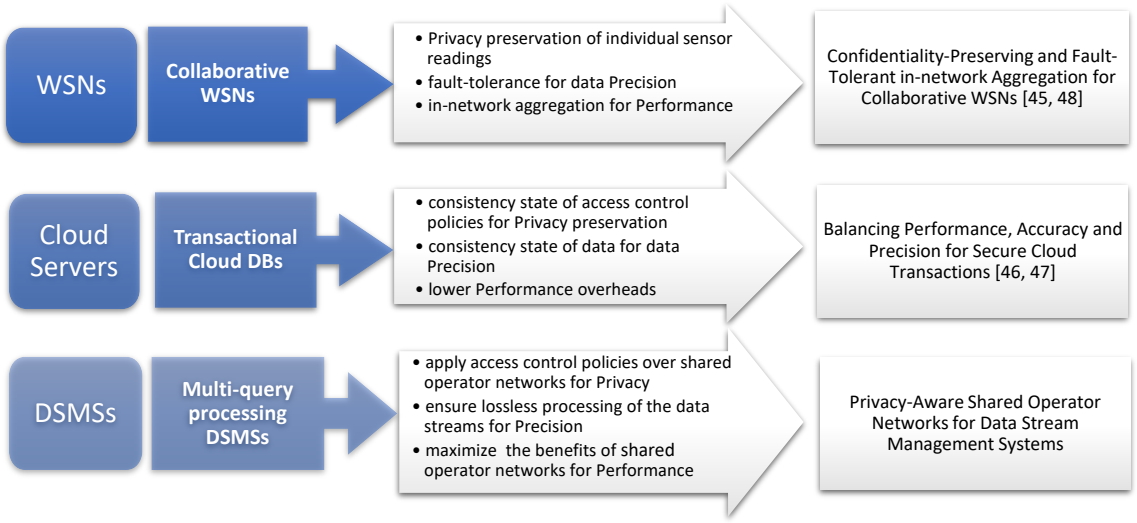


Figure 2: Dissertation contributions towards each representative distributed system

Figure 2 outlines the contributions of this dissertation with respect to each class of distributed systems. The diagram presents the general classes of distributed systems on the left, then moving right, the classes narrow down to specific applications within each class. Further to the right, the arrows list the solutions presented by this dissertation to balance the 3Ps in each application, then finally, to the right the protocols designed for each application are listed.

1.3 ROADMAP

The rest of this dissertation is organized as follows. Chapter 2 presents necessary background information on the three representative distributed systems discussed in the remaining of

the dissertation and a detailed discussion of the 3Ps. Chapter 3 presents the design and implementation of the privacy-preserving fault-tolerant protocol for in-network aggregation in WSNs. Chapter 4 describe the design and implementation of the data and access control policies consistency models in cloud databases. Chapter 5 discusses the solution presented for achieving privacy preservation in shared operator networks in DSMSs. Chapter 6 discusses related work and alternative approaches while Chapter 7 concludes the dissertation.

2.0 DISTRIBUTED SYSTEMS BACKGROUND

This chapter sets the stage for the rest of this dissertation by describing the properties of the three different classes of distributed systems explored in this dissertation. A detailed discussion of the unique features and constraints of each class of applications is presented. Section 2.1 discusses the challenges and constraints found in designing reliable and efficient wireless sensor network applications. Section 2.2 discusses the general design goals and requirements of cloud servers and the services they provide. Section 2.3 presents the constraints and challenges found in designing data stream database management systems. Finally, Section 2.4 presents a discussion of how the three distributed systems presented in this dissertation cover a broad range of interesting classes of distributed systems and the challenges associated with implementing the 3Ps within each class of distributed systems.

2.1 WIRELESS SENSOR NETWORKS

Wireless Sensor Networks (WSNs) are collections of tiny devices capable of sensing, computing, and wirelessly communicating to monitor and control events of interest or certain phenomenons in a distributed manner. These tiny sensor nodes are densely deployed either inside the environment containing the phenomenon being sensed or very close to it. Depending on the environment, different types of WSNs can be deployed. Some examples of the different types of networks include: terrestrial, underground, underwater, multimedia and mobile WSNs. One unique and common feature of all the different types of sensor networks is the cooperative effort of sensor nodes. Instead of sending the raw sensed data to the nodes responsible for the data fusion, sensor nodes use their processing abilities to locally carry

out simple computations and transmit only the required and partially processed data [14]. Sensors can also integrate the data they receive from nearby sensor nodes and transmit aggregate results on behalf of a group of sensors [56, 129].

The above described features and types of WSNs ensure a wide range of applications. Some of the application areas are health care, military, security and building surveillance, and environment and pollution monitoring. For example, the physiological data about a patient can be monitored remotely by a doctor. While this is more convenient for the patient, it also allows the doctor to better understand the patients current health conditions. Sensor networks can also be used to detect foreign chemical agents in the air and the water. They can help to identify the type, concentration, and location of pollutants. In essence, sensor networks will provide the end user with intelligence and a better understanding of the environment. It is envisioned that wireless sensor networks will continue to be an integral part of life more than the present-day personal computers [115].

Each one of the previously mentioned applications has its own network, hardware and deployment requirements. However, no matter the kind of the application, the communication scenario is always identical. A user sends several queries to the network. The set of nodes that is specified by the query, activate their sensors and start collecting raw data from the physical environment (e.g., temperature, humidity, etc.). The sensed data will start flowing back to the sink nodes where several computations may be performed internally by aggregator sensor nodes or by the final sink node. The sink finally sends back the query answer to the user.

2.1.1 CONSTRAINTS AND CHALLENGES

Most WSNs have the same set of constraints and general requirements. The design of sensor networks is influenced by many factors, which include *fault tolerance*; *hardware constraints*; *power consumption*; and *data privacy*. This section will present more details on some of these constraints.

One of the primary constraints of sensor nodes is the low power consumption requirement. In some application scenarios, replenishment of power resources in sensor networks might be

impossible. Sensor node lifetime, therefore, shows a strong dependence on battery lifetime. In a sensor network, each node plays the dual role of data originator and data router. The malfunctioning of few nodes can cause significant topological changes and might require re-routing of packets and re-organization of the network. This also affects the reliability of collected data and causes further fault-tolerance issues. Hence, power conservation and power management take on primary importance in the design of these networks. It is for these reasons that sensor networks must operate power-aware protocols and algorithms.

Another major challenge in the design of WSNs is the problems associated with sensor nodes failures. In applications where sensors are deployed in hostile environments, some sensor nodes may fail or be blocked due to lack of power, have physical damage or environmental interference. Omission failures due to weak communication links are also very common. The failure of sensor nodes should not affect the overall functioning of the sensor network. This is the reliability or fault tolerance challenge [83]. Fault tolerance is the ability to sustain sensor network functionalities without any interruption due to sensor node failures [77]. Therefore, WSNs must have inbuilt mechanisms that can accommodate such failures and ensure high functional sensor networks and accordingly reliable and precise data collection.

In addition to the above challenges, WSNs have also thrust privacy concerns to the forefront. Sensor networks interact closely with their physical environments and with people and different stakeholders, posing a lot of confidentiality and privacy problems. Consider WSNs applications that involve surveillance or monitoring of properties, work places or patients vitals. These networks are constantly sensing and reporting sensitive information. Adversaries within the proximity of the network could infiltrate the network, eavesdrop, and gain useful information.

2.1.2 RESEARCH TO DATE

The majority of research concerning WSNs look for solutions to realize one or two of the challenges discussed in the previous section simultaneously. That is, either the trade-offs between privacy and performance or precision and performance were studied at a time.

Some research efforts that focus on preserving power and energy consumptions in WSNs can be found in [153, 126, 75, 60, 73, 151]. In-network data aggregation [56] has been studied as an energy-efficient and bandwidth preserving process that moves the integration and filtering of sensor data into the network itself. In-network aggregation allows each node along the data routing path to aggregate *all* values received from its children into a *single* response value (thereby avoiding the transmission of messages from each sensor to the data sink). However, different schemes for in-network aggregation impose different challenges. For example, *tree-based* in-network aggregation provides the minimal communication overhead by using a spanning tree across all sensor nodes, but a single link failure in this model leads to the loss of all data from the subtree connected by that link. Given that WSNs are characterized by high rates of communication failures (up to 30% loss rate [156]), this approach can lead to large errors in the average case.

On the other hand,*multipath-based* in-network aggregation approaches add robustness to the traditional tree structure by taking advantage of the broadcast medium that WSNs typically use in communications. However, multipath aggregation techniques must be carried out carefully to avoid overcounting when computing aggregates. This is particularly important in duplicate sensitive functions such as SUM and AVG [24, 25].

When the security and privacy concerns started rising up in WSNs applications, researchers started looking for solutions to these privacy problems. All research efforts were attempts to fix the privacy problems by adding on security features to already existing sensor networks. As a result, and because sensor networks pose unique challenges, traditional security techniques used in traditional networks, such as public-key cryptographic systems, could not be applied directly due to their high computational power requirements. Similarly, existing mechanisms for carrying out confidential in-network aggregation either require the use of expensive cryptographic primitives that are unsuitable for use in the resource-limited sensor environments (e.g., [96]), or assume perfectly reliable communication links (e.g., [19, 116]) and therefore do not achieve any fault-tolerance.

Since the 3Ps properties were never thought of as part of the design process of wireless sensor networks, all the above research efforts has never achieved the desirable balance between these properties. The work presented in Chapter 3 of this dissertation contributes

to the design of wireless sensor networks by investigating algorithms and protocols that are capable of balancing all 3Ps properties.

2.2 CLOUD SERVERS

Cloud Computing has emerged as a computing paradigm that can be viewed as a virtualized pool of computing resources (e.g., storage, processing power, memory, applications, services, and network bandwidth), where customers provision and de-provision resources as they need. Cloud computing represents the vision of providing computing services as public utilities like water and electricity. Resources such as storage and computations can be outsourced from organizations to next generation data centers hosted by companies such as Amazon, Google, Yahoo, and Microsoft. Such companies help free organizations from requiring expensive infrastructure and in-house expertise, and instead make use of the cloud providers to maintain, support, and broker access to high-end resources. From an economic perspective, cloud consumers can save huge IT capital investments and be charged on the basis of a pay-only-for-what-you-use pricing model.

The cloud community has extensively used the following three service models to categorize the cloud services [103]:

Software-as-a-Service(SaaS). The most widely used model of cloud services is the SaaS model in which the customers have access to the applications running on the cloud providers infrastructure. Cloud consumers do not have control over the cloud infrastructure that often employs a multi-tenancy system architecture, namely, different cloud consumers' applications are organized in a single logical environment on the SaaS cloud to achieve economies of scale and optimization in terms of speed, security, availability, disaster recovery, and maintenance. Examples of SaaS include SalesForce.com, Google Mail, Google Docs, and so forth.

Platform-as-a-Service(PaaS). PaaS is a development platform supporting the full "Software Lifecycle" which allows cloud consumers to develop cloud services and applications (e.g., SaaS) directly on the PaaS cloud. Hence the difference between SaaS and PaaS is that SaaS only hosts completed cloud applications whereas PaaS offers a development platform

that hosts both completed and in-progress cloud applications. This requires PaaS, in addition to supporting application hosting environment, to possess development infrastructure including programming environment, tools, configuration management, and so forth. An example of PaaS is Google AppEngine [5].

Infrastructure-as-a-Service(IaaS). The IaaS model enables customers to rent and use the providers resources (storage, processing, and network). This model greatly minimizes the need for huge initial investment in computing hardware such as servers, networking devices and processing power. IaaS and other associated services have enabled startups and other businesses focus on their core competencies without worrying much about the provisioning and management of infrastructure. Virtualization is extensively used in IaaS cloud in order to integrate/decompose physical resources in an ad-hoc manner to meet growing or shrinking resource demand from cloud consumers. The basic strategy of virtualization is to set up independent virtual machines (VM) that are isolated from both the underlying hardware and other VMs. An example of IaaS is Amazon's EC2 [1].

These cloud service models are designed with elasticity as the common design aspect. Elasticity provides users an illusion of infinite, on-demand resources [18] making it an attractive environment for highly-scalable multi-tiered applications. As appealing as this may sound to many applications, a lot of key challenges exist that are setting boundaries to a wider adoption of the cloud [52]. Security, privacy, data integrity and consistency are some of the most significant barriers. There is a lot of uncertainty about how security at all levels (e.g., network, host, application, and data levels) can be achieved. Furthermore, the implications of the added security and data integrity measures on the overall cloud systems performance is still unclear in many applications. The following section introduces some of the cloud challenges that pertains to one of the most commonly used cloud services which is cloud databases.

2.2.1 CLOUD DATABASE REQUIREMENTS AND CHALLENGES

Cloud databases are one of the services that cloud servers offer to its users. It serves many of the same functions as traditional database management systems with the added flexi-

bility and scalability of cloud computing. Two cloud database environment models exist: traditional and database as a service (DBaaS). Traditional database cloud model requires companies to provision all of the underlying infrastructure and resources necessary to manage their databases on-premises with cloud-based technology. On the other hand, the DBaaS is a database cloud service that takes over the management of the underlying infrastructure and resources cloud databases require and allow companies to take advantage of services in the cloud.

DBaaS model offers an array of distinct advantages to its users, from the elimination of the physical infrastructure and its associated costs, to the benefits of instantaneous scalability, elasticity, and fault-tolerance support. Despite these advantages, there are some challenges that need to be considered when using databases on cloud servers.

The first set of challenges arise from both the elasticity and scalability requirements of cloud servers. These requirements can create multiple issues for back-end transactional database systems, which were designed without elasticity in mind. Despite the efforts of key-value stores like Amazon's SimpleDB [35], Dynamo [51], and Google's Bigtable [40] to provide scalable access to huge amounts of data, transactional guarantees remain a bottleneck [50].

To provide scalability and elasticity, cloud services often make heavy use of replication to ensure consistent performance and availability. The mechanisms used for data replication vary according to the nature of the data; more copies are needed for critical data and they are replicated on multiple servers across multiple data centers. On the other hand, non-critical, reproducible data are stored at reduced levels of redundancy. The pricing model is related to the replication strategy.

As a result of the heavy use of data replication, many cloud services rely on the notion of *eventual consistency* when propagating data throughout the system. This consistency model is a variant of weak consistency that allows data to be inconsistent among some replicas during the data update process, but ensures that updates will eventually be propagated to all replicas.

The implications of this consistency model on transactional databases are very serious. It becomes very difficult to strictly maintain the database ACID guarantees [69], as the 'C' (consistency) part of ACID is sacrificed to provide reasonable availability [12].

Another set of challenges that can arise in transactional database systems that are deployed in the cloud is that data access control policies would typically be replicated—very much like data—among multiple sites, often following the same weak or eventual consistency model. In systems that host sensitive resources, accesses are protected via authorization policies that describe the conditions under which users should be permitted access to resources. These policies describe relationships between the system principals, as well as the certified credentials that users must provide to attest to their attributes. With the replication of the access control policies under the eventual consistency models, it becomes possible for a policy-based authorization system to make unsafe decisions using stale policies.

It is unarguable that, the safety of transactional database systems deployed in cloud environments are being put at risk due to the following reasons. First, the systems may suffer from *policy inconsistencies* during policy updates due to the relaxed consistency model underlying most cloud services. For example, it is possible for several versions of the policy to be observed at multiple sites within a single transaction, leading to inconsistent (and likely unsafe) access decisions during the transaction. Second, it is possible for external factors to cause *user credential inconsistencies* over the lifetime of a transaction [91]. For instance, a user's login credentials could be invalidated or revoked after collection by the authorization server, but before the completion of the transaction, which could also lead to unsafe access decisions. Finally, the data inconsistencies can cause serious data precision problems. For instance, consider transactional databases with patients information deployed on the cloud and are replicated under the eventual consistency models, imprecise physicians query results could be generated jeopardizing patients' health conditions.

2.2.2 RESEARCH TO DATE

Most research efforts in the cloud computing field focus on the privacy and integrity concerns of data outsourced to the cloud. Some other research has only targeted performance issues of the cloud computing. As a result, the trade-offs of balancing 3Ps in cloud servers remain to be a widely unexplored research area. This section explores some of the research efforts that focused on balancing either privacy with performance or precision with performance.

Most of data privacy and integrity issues in the cloud became of more concern after the widespread use of cloud servers. Many clients indicated the need for efficient techniques to verify the integrity of their outsourced data with minimum computation, communication, and storage overhead. Consequently, many researchers have focused on the problem of efficient Provable Data Possession (PDP), which are considered two of the 3Ps properties, and proposed different schemes to audit the data stored on remote servers. Simply, PDP is a technique for validating data integrity over remote servers. Ateniese et al. [20] have formalized a PDP model. In that model, the data owner pre-processes the data file to generate some metadata that will be used later for verification purposes through a challenge response protocol with the remote/cloud server. The file is then sent to be stored on an untrusted server, and the owner may delete the local copy of the file. Later, the server demonstrates that the data file has not been deleted or tampered with by responding to challenges sent from the verifier. Later, researchers have proposed different variations of PDP schemes under different cryptographic assumptions [155, 127].

Mykletun et al. [107] provide mechanisms for outsourced databases that assure the querier that the query results have not been tampered with and are authentic (with respect to the actual data owner). They investigate both the trade-offs of both the privacy and the performance aspects of the problem and construct several secure and practical schemes that facilitate integrity and authenticity of query replies while incurring low computational and communication costs.

More security and privacy challenges arise in the cloud due to the multitenancy architecture of the cloud servers that allows virtualization of the underlying hardware and software resources. In an experiment to show the criticality of this problem, Thomas Ristenpart et al. [120] used engineering techniques to infer the virtualized resource allocation strategy from cloud servers and successfully placed their virtualized application instance on the same physical machine as the target victim. They were then able to extract the victims private information through traffic patterns and other side-channel information. Therefore, multitenancy security and privacy poses one of the critical challenges for the public cloud. However, little work exists that addresses these privacy problems in an efficient manner to maintain the scalability of the cloud computing environment.

There has been some research efforts that study the interplay between data precision and consistency and performance in cloud servers. Hiroshi Wada et al. [145] reported interesting experimental results when they examined the experience of cloud storage consumers in regard to weak consistency and possible performance tradeoffs to justify it. Their results showed that platforms differed widely in how much weak consistency is seen by consumers. For example, on Amazon SimpleDB, the consumers read requests experienced frequent stale reads and inter-item inconsistencies. While on other platforms, inconsistencies seemed to be very rare; perhaps only happening if there is a failure of one of the nodes or communication links actually used in the computation. They also observed that from the clients perspective there was not much performance gains when issuing weak-consistency reads versus consistent reads. While there may be benefits to the cloud provider when eventual consistent reads are done, but these gains seem not to be passed on to the consumers.

Other research efforts focused on allowing cloud applications to declare their own consistency and latency priorities via consistency based service level agreements (SLAs) [138] or by having an application-based adaptive consistency strategies in which applications select the most suitable strategy automatically at runtime [146].

These results only prove that there is not any certainty on the consistency levels of data given by any of the cloud providers which makes the criticality of the inconsistent states of access control policies on the cloud servers more apparent. Chapter 4 of this dissertation highlights this problem and proposes different consistency solutions to achieve a balance between all 3Ps in transactional cloud database systems.

2.3 DATA STREAM MANAGEMENT SYSTEMS

In today's cyber world, the ubiquity of sensing devices as well as mobile and web applications leads to the generation of huge amounts of data, which take the form of streams. Sensing devices such as wired or wireless sensors are used in several monitoring applications such as weather observation and environment monitoring in general, health monitoring, monitoring of assembly lines in factories, RFID monitoring, or road monitoring, and they all generate

huge amounts of data. Web applications such as stock price analysis or network traffic monitoring can produce even millions of samples per second. Social networks, and the Internet of Things [22] are other applications and devices that generate data streams.

These applications share characteristic properties which are especially challenging for traditional data management systems. First of all, the sensed data is produced at a very high frequency, often in a bursty manner, which may pose real-time requirements on processing applications and may allow them only one pass over the data. Second, the data is not only produced rapidly, but also continuously forming a data stream. Data streams can be unbounded, i.e., it is not clear, when the stream will end.

Data Stream Management Systems (DSMSs), as a new paradigm, have been implemented to deal with all of these stream data issues [62]. They have become the popular solutions to handle data streams, by efficiently supporting continuous queries (CQs), which process data as they arrive on the fly. In contrast to common data management systems, and based on the nature of stream applications, DSMSs have to react to incoming data and deliver results to users frequently. In DSMSs, users register CQs in the system once and after that the queries are executed incessantly. A CQ produces new results as long as new data arrives at the system and is continuously executed until it is explicitly uninstalled from the system.

To process CQs, query execution plans are generated. Each plan consists of a number of relational operators, such as "select" , "project", "join" and "aggregate" operators. A query execution plan is composed of an interconnection of these operators to yield the desired output of the query, such that the operators in a query plan may be regarded as nodes in a network. The operators act on data elements as they arrive and cannot assume the data stream to be finite.

Query operators are classified as stateless and stateful operators. A stateless operator, such as "select" produces an output tuple based solely on the current input tuple. Conversely, a stateful operator, such as "join" or "aggregate", needs to refer to values from previous input tuples. Due to the fact that input streams are infinite, DSMSs use either tumbling or sliding windows, to limit the state of operators. Sliding windows allow the output to be continuously computed based on the most recent portion of the stream data. In addition, a sliding window is specified through a length (or range), and a slide, which can be either time interval or

tuple count. These two types of windows are called *time-based* and *tuple-based windows*, respectively [23].

Now, after many years of research on continuous queries and data streams, many research prototypes have evolved [13, 104, 29, 43] and have gained a great momentum in many stream-based applications. Many of these research efforts have turned into mature industry solutions by the big players in the field of data management, such as Microsoft [8], IBM [7], or Oracle [9]. However, despite these efforts, there are still many issues and questions that remain open for further investigation and research. The following sections, will highlight some of these issues and briefly review some of the existing research in DSMSs.

2.3.1 DSMS CONSTRAINTS AND CHALLENGES

There are several challenges and constraints when it comes to designing efficient DSMSs [23, 65]. Since DSMSs are typically used for handling large amounts of data arriving with high velocity, enterprises move to Cloud Infrastructures to minimize the purchase and maintenance costs of machinery, and be able to scale their services to handle the large volumes of data. Furthermore, to increase DSMSs throughput in processing multiple queries simultaneously and minimize the memory usage, computation times, and other cloud costs, DSMSs tend to optimize the execution of the continuous queries. To achieve this, many DSMSs perform multi-query optimizations that can generate a combined evaluation plan for a collection of multiple queries. Multi-query optimizations take advantage of the fact that different queries may share common data streams, share overlapping query predicates, or calculate similar aggregate operations. They use algorithms that can identify and compute the common sub-expressions between queries once and reuse them to produce low-cost shared-operator networks execution plans. The idea of sharing partial query results is typically less expensive compared to a serial execution of queries.

With most of the research in DSMSs being focused on query processing and optimization [13], distribution [44] and integration of data sources [11], some of the security and privacy issues in DSMSs were overlooked for many years. In DSMSs that host sensitive data, e.g., patient monitoring data, financial data, etc., data providers restrict data accesses

via access control policies that describe the conditions under which users are permitted access to specific data streams. Accordingly, the continuous queries registered by the system users may be granted or denied access to specific data streams during the execution of these queries. Unfortunately, traditional privacy-preservation mechanisms, which typically assume finite persistent datasets and system-centric access control policies, become largely inapplicable in this new stream paradigm. The reason is that unbounded queries will, by definition, run long enough to experience frequent changes in their access control and/or changes to the "sensitivity" of the streamed data during their run.

The intersection of the privacy and performance requirements in DSMSs leads to the question of whether these two requirements can be balanced without compromising the data precision and the quality of the queries output. Given the interleaved execution of queries in DSMSs, managing access control per user or group of users over shared operator networks in an efficient manner without affecting the query output precision becomes a challenging task. This dissertation addresses the balancing of the 3Ps in DSMSs in Chapter 5.

2.3.2 RESEARCH TO DATE

DBMSs and DSMSs share some of their query optimization goals, both try to minimize computational costs, memory usage and size of intermediate results stored in main memory. But obviously, the priorities for these goals are different for the two systems. DBMS mainly tries to reduce the costs of disk accesses [54], while DSMS mainly have to reduce memory usage and computation time to be fast enough. Of course, these different goals stem from the different data handling strategies (permanent storage vs. real-time processing). In a DSMS a query is also not only optimized before execution, but it is adaptively optimized during its run-time. This enables the system to react to changes of input streams and system and network conditions.

As mentioned in the previous section, there has been plenty of research efforts that focused on optimizing the execution of the continuous queries in DSMSs [132, 157, 130, 70]. These research efforts focus on ways of interleaving the execution of multiple continuous queries. Execution plans of registered queries are combined into one big plan to reuse results

of common operators for multiple queries to optimize the memory and CPU cycles required in processing the queries and increase the overall systems throughput.

Adaptive techniques for query processing has been studied in several research. Madden and Franklin [100], focus on query execution strategies over data streams generated by sensors, and Madden et al. [102] discuss adaptive processing techniques for multiple continuous queries. The Aurora project [13] performs both compile-time optimization (e.g., reordering operators, shared state for common subexpressions) and run-time optimizations. As part of run-time optimization, Aurora detects resource overload and performs load shedding based on application-specific measures of quality of service.

On the other hand, most research efforts that address privacy issues in DSMS using access control of the data streams use one of two techniques. They either apply access control over operator networks before or after the execution of the network. These techniques are referred to as pre-filtering and post-filtering, respectively [110]. In both techniques, the fixed placement of the access control filters may considerably limit query performance. Pre-filtering means cutting off the streams in case any of the users lose access, which will affect the input streams feeding into the shared operator networks. Post-filtering on the other hand, causes operator networks to execute until the end and then denying access to the query results for the users that lost access. It is obvious that this technique is both inefficient and wasteful to systems' resources.

There has been no research to date that allows access control policies on the input data streams and queries to be applied over shared operator networks to maintain both privacy and high performance without affecting the query results precision. For this reason, this dissertation focuses on balancing all 3Ps in DSMSs. The contributions of this dissertation in this area will be presented in Chapter 5.

2.4 3PS DISCUSSION

As shown in this chapter, different types of distributed systems are very diverse in their nature, characteristics and the constraints under which they operate with. Despite this

diversity, several distributed systems demonstrate the need to incorporate all 3P properties in their design in a balanced way. Many distributed systems have security concerns of the data they handle and protecting users privacy is vital to promote the widespread use of these systems. Accuracy and precision of the data is another important property to distributed systems. Without data precision guarantees, users will less likely be motivated to use the services offered by the distributed systems. And finally, there is less benefit to a distributed system if it can achieve high privacy and precision and yet is unable to meet high performance standards.

This dissertation focuses its efforts on designing distributed systems that can achieve a good balance between all 3Ps properties. It will be shown in the remainder of this dissertation that achieving all 3Ps is not simply a matter of implementation and deployment of some techniques and protocols, but it is a process that requires careful consideration of the system constraints and limitations. If these properties are to be accomplished in a balanced manner in distributed systems, the systems-level constraints must be studied carefully and these properties must be well defined with respect to the individual unique characteristics of each distributed system.

The three representative distributed systems studied by this dissertation cover up a broader range of classes of distributed systems where each class has a distinctive set of constraints and requirements for the 3Ps. Table 1 summarizes the different constraints and requirements of each class of applications. WSNs are distributed systems that are classified as highly resource constrained systems. All resources in these systems from energy, power, computational, and bandwidth resources are very sparse. Hence, any algorithms or protocols designed for these systems need to be very resource-aware (i.e., lightweight). Cloud computing systems are highly dynamic and scalable systems. Despite that cloud resources are very abundant, unlike WSNs, with these plentiful resources comes the issues of maintaining high systems availability, elasticity and scalability with high performance requirements. Finally, the last class of distributed systems are the real-time performance and processing systems demonstrated in this dissertation through DSMSs. These distributed systems put on high emphasis in their design on real-time data processing due to the high volumes and high input rates of the input data managed and handled by these systems.

Distributed System	Constraints and requirements
Wireless Sensor Networks (WSNs)	Highly resource constrained DSs • Energy constraints • Bandwidth constraints • Computational constraints • Deployment and coverage constraints
Cloud Systems	Elastic and dynamic DSs • High availability requirements • High scalability requirements • High performance requirements • Multi-tenancy requirements
Data Stream Management Systems (DSMSs)	Real-time constrained DSs • High input data rates • Real-time processing requirements • Computing resources constraints

Table 1: Constraints and requirements of the three classes of representative systems

2.4.1 CHALLENGES AND GOALS REVISITED

Because the 3Ps properties span a vast array of descriptions, taking shape in many different forms depending on the properties, constraints and requirements of each distributed system, they must be well defined in terms of each distributed system independently. Therefore, it becomes clear that the 3Ps are not *absolute* properties. Their definitions and expectations differ from one distributed system to another.

In this section, the concrete challenges and constraints of an example application within each of the three representative systems studied by this dissertation will be listed in details along with the design goals that this dissertation will address for each of these applications.

From the background information that was presented in Section 2.1 on WSNs, this dissertation identifies the need for balancing the 3Ps properties in a category of WSN applications

that require the privacy preservation of individual sensor readings while being able to perform in-network aggregation of those readings. The following are the 3P goals that need to be balanced while designing this system:

- **Goal 1:** Protect individual sensor readings from external or internal system eavesdroppers and malicious system insiders using lightweight encryption techniques that would suit the limited computational powers of sensors and induce minimal energy consumption overheads. [*Privacy-Performance*]
- **Goal 2:** Apply fault-tolerance mechanisms to the in-network aggregation process to tolerate communication link failures and ensure the reliable delivery of aggregate values while preserving the network bandwidth and power consumption. [*Precision-Performance*]
- **Goal 3:** Identify the best suitable homomorphic cryptosystems that can be used to provide in-network aggregation of encrypted data. [*Privacy-Performance*]

The following are the design challenges that are evident when dealing with WSNs:

- **Challenge 1:** Traditional data encryption techniques for privacy preservation perform expensive computations that might not suit the resource limited nature of WSNs and they can not be used if data aggregation is required.
- **Challenge 2:** In-network aggregation is a cost-effective way of aggregating sensor readings, but on the other hand, has no guarantees on the precision of the final aggregate results (due to high communication loss rates or over-counting values).
- **Challenge 3:** Most sensors have limited energy supply and very limited computational powers which constraints the protocols and algorithms that can be executed in these networks.

Next, this dissertation identifies serious privacy problems that can occur in transactional cloud database applications when access control policies are replicated and updated using the weak consistency models of the cloud servers as discussed in Section 2.2. Balancing the 3Ps requirements in this category of distributed systems is a non-trivial task. High consistency guarantees can hurt systems' performance, yet low consistency models can cause unsafe access decisions and hence compromise the systems' privacy. Accordingly, the following is a

list of the 3P design goals for policy-based authorization systems of transactional databases in the cloud:

- **Goal 1:** Ensure no unsafe data access decisions are made at any point of time during transactions execution. [*Privacy*]
- **Goal 2:** Assure that data reads and writes during transactions execution at different cloud servers fall into some degree of approved consistency level. [*Precision*]
- **Goal 3:** The consistency measures taken to achieve both data privacy and precision must not affect the cloud availability and overall performance. [*Performance*]
- **Goal 4:** Provide a broad range of consistency levels for both data and policies that can be used by a wide range of applications based on their consistency and performance requirements. [*all 3Ps*]

Balancing all 3Ps in these applications need to be considered given the following challenges of transcational database systems executing on cloud servers:

- **Challenge 1:** Data privacy can be compromised due to the weak (eventual) consistency model of cloud servers and the long time span of executing transactional databases. Systems may suffer from *policy inconsistencies* and/or *user credential inconsistencies*.
- **Challenge 2:** Transactional databases may execute over long periods of time in which data may fall in an inconsistent state causing potential data precision problems.
- **Challenge 3:** Applications executing on the cloud that have high availability and fast response time constraints can not afford the potential performance overheads of executing strict consistency models.

Finally, this dissertation identifies the need for balancing the 3Ps in applications that execute continuous queries governed by access control policies in DSMSs. The following are the 3Ps design goals:

- **Goal 1:** Enforce access control policies defined by data providers over their data streams while allowing for shared-operator networks to execute in an undisruptive manner. [*Privacy*]

- **Goal 2:** Processing continuous queries in DSMSs should not be interrupted due to intermittent changes in access control policies or users' credentials. Gain or loss of access should not cause the loss of data tuples in processing queries. [*Precision*]
- **Goal 3:** Access control enforcement mechanisms should not affect the real-time constraints of the system (i.e., queries should not face any delays in their processing) and should be handled in a cost-effective way. [*Performance*]

The challenges in meeting the 3Ps requirements in this system are summarized as follows:

- **Challenge 1:** Sharing partial results through interleaved query execution and shared-operator networks is an important performance optimization for DSMSs and needs to be maintained.
- **Challenge 2:** Traditional techniques of enforcing access control over operator networks are either in-efficient or not applicable over shared-operator networks.
- **Challenge 3:** Enforcing access control over shared executing queries can cause disruption in the execution of operator networks which in turn can have severe performance overheads and data imprecision problems.

From the design challenges listed for each application in each of the representative distributed system, the reason for why these systems were chosen by this dissertation becomes evident. Each system operates within a given set of constraints and design challenges that makes balancing all 3P properties an interesting problem to investigate. Therefore, these systems are a good representation of a broader class of distributed systems applications that share similar set of challenges and constraints.

The remainder of this dissertation seeks to facilitate the design, implementation, analysis, and optimization of the algorithms presented to meet the design goals of each system. Specifically, the dissertation addresses the gaps that exist between the common implementations and designs of these systems that have been adopted in real-life for many years and the lack of a balanced implementation of the 3Ps in them.

3.0 PRIVACY-PRESERVING AND FAULT-TOLERANT IN-NETWORK AGGREGATION FOR COLLABORATIVE WSNS

This chapter describes the design and implementation of the first representative distributed system application of this dissertation: privacy-preserving in-network aggregation protocol for Collaborative WSNs (CWSNs) [80] and [79]. The proposed protocol allows the in-network aggregation of sensor readings to be performed in a secure, fault tolerant and efficient manner.

In this system, the 3Ps requirements are defined as follows:

- **Privacy**: individual sensor readings and their aggregate values are not revealed to any external (eavesdropper) or internal (compromised sensor) attacker, only to a trusted base station.
- **Precision**: demonstrated through two properties:
  (i) Robustness and fault-tolerance: sensor readings that are lost due to link errors are compensated for *at most once*.
  (ii) Exact aggregation: Instead of providing probabilistic query aggregate results (e.g., [46, 61]), the proposed approach provides an exact aggregate result in case of no link failures. With link failures introduced, the final aggregate deviates by exactly the lost value and not by some derivative of that value.
- **Performance**: Privacy preservation and fault tolerance should be achieved with minimal overheads on the size of packets transmitted and amount of computation required. This property is vital due to the resource-constrained nature of sensor systems. The protocol presented induces low overheads on the size of packets transmitted and amount of computations required.

The proposed protocol in this chapter achieves a desirable balance between the 3Ps requirements by making the following contributions:

- A protocol is proposed to ensure the end-to-end privacy of the individual readings while allowing the in-network aggregation process to be performed on the encrypted readings using symmetric-key homomorphic cryptosystems.
- Fault-tolerance of the sensor networks is achieved by using a multipath aggregation protocol that is capable of detecting and recovering from a variety of intermittent link-level failure scenarios while carefully handling duplicate sensitive aggregates. This ensures that with high probability that every sensor reading contributes to the final aggregate at most once (to avoid duplicates).
- Formal proofs and analysis for the privacy and correctness properties of the proposed algorithm are presented.
- The communication and energy consumption overheads associated with the protocol are evaluated through simulations.

The rest of this chapter is organized as follows. Section 3.1 discusses some common applications for WSNs. Section 3.2 discusses the network and attack models, as well as the basic building blocks for the new protocol. Section 3.3 describes the confidentiality-preserving and fault-tolerant in-network aggregation protocol in details. Section 3.4 proves the security and correctness of the proposed protocol and Section 3.5 presents the experiments and evaluations of data transmission and energy consumption overheads. Finally, Section 3.6 summarizes and concludes the work presented in this chapter.

3.1 WSNS MOTIVATING APPLICATION

With the development of sensor, wireless mobile communication, embedded system and cloud computing, the technologies of Internet of Things (IoT) have been widely used in logistics, smart meter, public security, intelligent buildings and so on. Because of its huge market prospects, IoT applications are regarded as the third wave of information technology

Figure 3: Collaborative Sensing over Shared Infrastructure (CSSI) example

after Internet and mobile communication networks. IoT applications are currently bridging between WSNs and the Internet to facilitate the remote management and control of the sensor networks [158], [92], [131], [87].

With the integration of WSNs and the Internet, this section will present an example application of WSNs and demonstrate through this application the need for balancing the 3Ps properties. In this example, WSNs collaboratively sense and detect phenomena of shared interest in intelligent building management applications usually referred to as *Collaborative Sensing over Shared Infrastructure (CSSI)* [95].

In these applications buildings or general purpose multi-purpose facilities may be equipped with sensing infrastructures that are owned and operated by different entities and stakeholders. In some situations, different stakeholders may not want to share information about the occupancy of individual rooms, although they might want to contribute in computing statistics about the occupancy of regions within the building in order to make better decisions about heating/cooling, public safety, facilities surveillance, and traffic monitoring.

To achieve this collaborative sensing in a privacy-preserving manner, sensors managed

by individuals or departments within the building could measure statistics like occupancy or temperature, encrypt their results, and forward the encrypted results through the shared sensing infrastructure. For performance and energy conservation, these readings are then aggregated *in encrypted form* as they are being sensed and sent within the network, thereby reducing overheads in the network. In the end, the aggregate value(s) are decrypted to derive the desired statistics. Figure 3 shows a representative diagram of a CSSI basic infrastructure.

These category of applications validate the need for achieving all the 3P properties. The first property, privacy, is demonstrated by the need of maintaining the privacy of individual sensor readings since the preservation of each sensor reading confidentiality ensures that information about user location privacy or the occupancy status of individual offices may not be disclosed. The second property, precision, is highly desired to ensure that the final aggregate values of the individual sensor readings are computed reliably to avoid any inaccurate decision makings based on false data collection. Finally, the algorithms and protocols executing on the sensor nodes have to be designed carefully to ensure high performance of sensor nodes and preserve the energy and bandwidth consumption of the entire network.

3.2 MODELS AND BUILDING BLOCKS

In Section 3.1 all 3Ps properties were identified in an example application of WSNs. Yet, in general, most WSNs applications demonstrate one way or another the need to balance all 3Ps properties. Securing the communication links between the sensor nodes to achieve privacy of individual sensor readings is an important property in many WSNs applications (think of medical health care military applications). The need for reliable communication links to maximize the precision of the aggregated sensor readings is another common property across all WSNs applications. The power and energy consumption constraints of WSNs are a general set of constraints in designing protocols and algorithms for WSNs applications to ensure high performance. Hence, the need to balance all 3P properties in WSNs becomes very evident in this class of highly constrained applications.

This section sets the stage for the rest of this chapter by describing the system model

of the privacy-preserving and fault-tolerant in-network aggregation protocol and presenting the main building blocks that were used to device this protocol. The network and attack model assumptions are presented as well as the cryptographic and routing primitives that the protocol builds upon.

3.2.1 NETWORK AND ATTACK MODEL

The network is assumed to operate as a multi-hop network that consists of n static sensor nodes and a single trusted sink node. Each sensor node has a unique identifier, ID, and shares a unique symmetric key with the sink. These keys are assumed to be pre-shared at deployment time, but could also be distributed through other means (e.g., [37, 36]). As usual, sensors are small, battery-operated devices capable of performing simple computations and wireless (broadcast) communications. The sink is a more capable node with higher computational and storage capabilities and no battery limitation. The sensors may belong to different stakeholders and execute collaborative sensing applications that use energy efficient in-network aggregation to compute statistics over the individual sensor readings.

The main security concern of this protocol is the data confidentiality with no bound on the number of attackers. Two types of attackers in the network are assumed:

(a) *Honest but curious* sensors that correctly perform the in-network aggregation process, but wish to learn information about the readings of other sensors if at all possible; this is representative of sensor deployments in which individual sensors may become compromised, but continue to function in a seemingly correct manner in order to gather information about the state of the larger network.

(b) *Quiet infiltrators* that are able to stealthily physically infiltrate the network, eavesdrop, and either accumulate the information gathered or send the information in an undetected way (e.g., using a different channel).

An adversary is assumed to control any arbitrary number of (colluding) attacker sensor nodes to extract information, and can eavesdrop on all communication channels. The sink node is assumed to remain uncompromised.

Relating back to the motivating example from Section 3.1, some sensor nodes that moni-

tor the occupancy of a facility could be compromised by attackers willing to gain information about the occupancy of specific rooms in the facility. The leak of such occupancy information could violate the privacy rights of the individuals within this facility. Attacks that attempt to falsify sensor readings, replay attacks and DoS attacks are out of scope of this work since these attacks can be addressed using the traditional security techniques and countermeasures such as access control, session identifiers, and firewalls. Also maintaining the privacy of the contributing sensor nodes is not a concern, that is, it is not the goal to hide the identity of those sensor nodes that contribute readings to a specific query.

3.2.2 CRYPTOGRAPHIC PRIMITIVES

Based on the attack model and the network assumptions described in the previous subsection, the choice of *end-to-end* encryption of sensor readings over *hop-by-hop* encryption was made for the following reasons:

- Using end-to-end encryption, intermediate nodes cannot decrypt the readings that they forward and thus data are not only protected from external eavesdroppers, but also from malicious compromised or curious nodes within the CWSNs.
- To aggregate data encrypted in a hop-by-hop manner, each sensor node must first decrypt each encrypted value that it has received, aggregate the resulting values, and then re-encrypt the aggregate. Hence, the overhead of the decrypt-aggregate-reencrypt operations of hop-by-hop increases linearly in the number of children of each node.
- End-to-end encryption eases the key distribution process by reducing the number of keys required by the system. Each sensor shares only one unique secret key with the sink node.

The privacy-preserving scheme makes use of the symmetric key, lightweight additively homomorphic stream cipher proposed in [33]. In this cryptosystem, a keyed pseudo-random generator is used to effectively generate keystreams that are used to encipher sensor readings stored as integer values. Encryption is simply addition mod M and decryption is subtraction mod M, where M is an upper-bound on the aggregate function to be computed. For example, a sensor node sharing a key k with the sink and using pseudo-random generator g can encrypt

its j^{th} reading of a value, v^j, as follows:

$$c^j = v^j + g^j(k) \bmod M$$

The sink can then recover the value v^j as follows:

$$v^j = c^j - g^j(k) \bmod M$$

A key feature of this cryptosystem is its ability to homomorphically combine values that are encrypted under the same or different keys. Consider two sensor nodes n_1 and n_2 sharing keys k_1 and k_2, respectively, with the sink (but not with each other). Suppose these principals wish to encrypt their i^{th} values v_1^i and v_2^i, respectively. The nodes encrypt their values as follows:

$$c_1^i = v_1^i + g_1^i(k_1) \bmod M \tag{3.1}$$

$$c_2^i = v_2^i + g_2^i(k_2) \bmod M \tag{3.2}$$

Given the aggregate value $C^i = c_1^i + c_2^i$, the sink can recover the aggregate key $K^i = g_1^i(k_1) + g_2^i(k_2)$ and decrypt the aggregate value $V^i = (v_1^i + v_2^i) = C^i - K^i \bmod M$. Note that neither v_1^i or v_2^i are disclosed via this process.

This symmetric key, additively homomorphic cryptosystem was proven to be semantically secure [33]. In the well-known notion of semantic security [67], the adversary is assumed to be a probabilistic polynomial time (PPT) Turing machine. In the model, the adversary can choose to compromise a subset of nodes and obtain all secrets of these nodes. With oracle access (i.e., assuming the adversary has the capability of querying a node and getting responses back), it can obtain from any of the uncompromised nodes the ciphertext for any chosen plaintext. Semantically secure cryptosystems demonstrate the adversary's inability to extract, in polynomial time, any information about the plaintext from a given ciphertext.

Another important property of this cryptosystem is the flexibility given in terms of which stream cipher to use as the pseudo-random keystream generator (g). That is, the homomorphic cryptographic system does not dictate any specific stream cipher, any stream cipher that is accredited by the cryptography community is a valid choice.

3.2.3 CASCADED RIDESHARING

Cascaded RideSharing was proposed in [64] as an efficient fault tolerance scheme for in-network aggregation using duplicate sensitive functions. The network faults or failures are assumed in links only and they are "omission" faults or "crash" faults. Cascaded RideSharing exploits the redundancy in the wireless medium to detect and correct communication link failures. To accomplish this, the sensor network is organized into a *track graph topology* [57]. Figure 4 illustrates a simple track graph topology. In such a topology, sensor nodes are organized in logical tracks, with the sink residing in track 0, sensors one hop away from the sink are in track 1, and so forth. A track consists of all nodes at the same distance from the sink. The aggregation path then forms a Directed Acyclic Graph (DAG) with multiple paths through the track graph, rather than a simple spanning tree. Each sensor node has one *primary* parent and one or more *backup* parents in the adjacent track towards the sink. Primary parents form a spanning tree that is used in case of error-free operations, while backup parents help compensate for errors in the primary links. Parents and multi-path routes are setup when the sink injects the query into the network.

In the RideSharing model, the parents of each sensor node are assumed to overhear each others transmissions. The nodes transmit their readings according to a predefined Time Division Multiple Access (TDMA) schedule [55]. TDMA scheduling allocates time slots for each sensor node depending on the topology and the node packet generation rates. TDMA protocols are power efficient since nodes in the network can enter inactive states until their allocated time slots. They also eliminate collisions and bound the delay. Figures 5 and 6 illustrate a simple aggregation spanning tree showing only the parent nodes and its corresponding TDMA scheduling. Note that with backup parents, the scheduling would look different since backup parents cannot be scheduled to transmit at the same time slots as primary parents.

Every sensor node transmits its reading to its primary parent during its allocated TDMA time slot. Note that not all sensors transmit data, only those that include new information, as determined by the application. For example, repeatedly transmitting the same temperature or other information would not add any new value, and therefore the application may choose

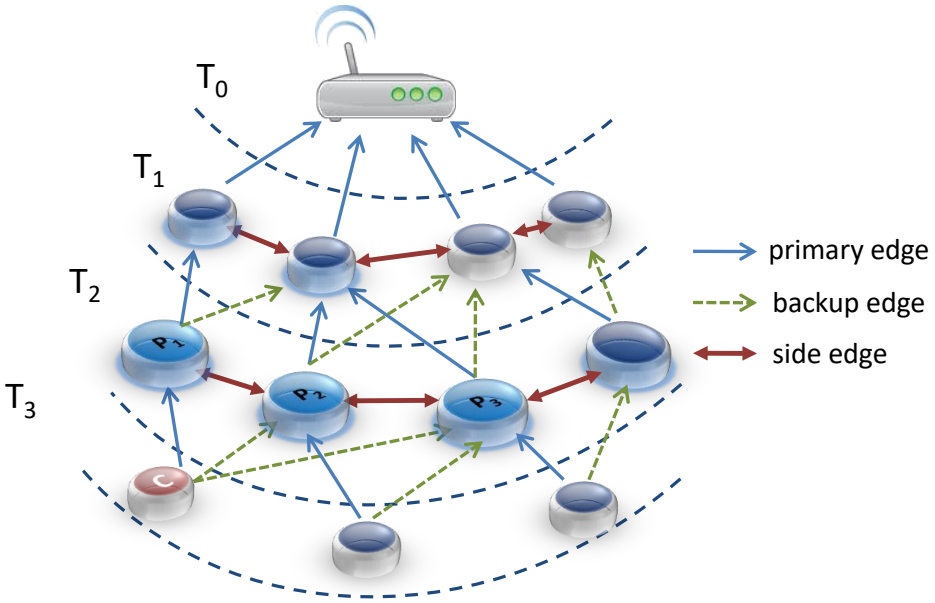


Figure 4: Track graph network topology

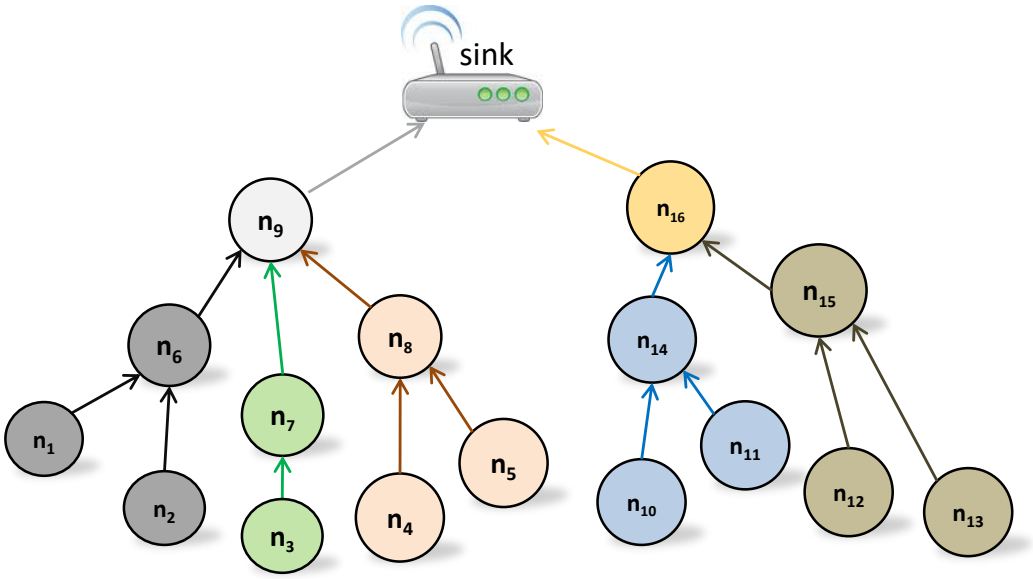


Figure 5: Primary parents spanning aggregation tree

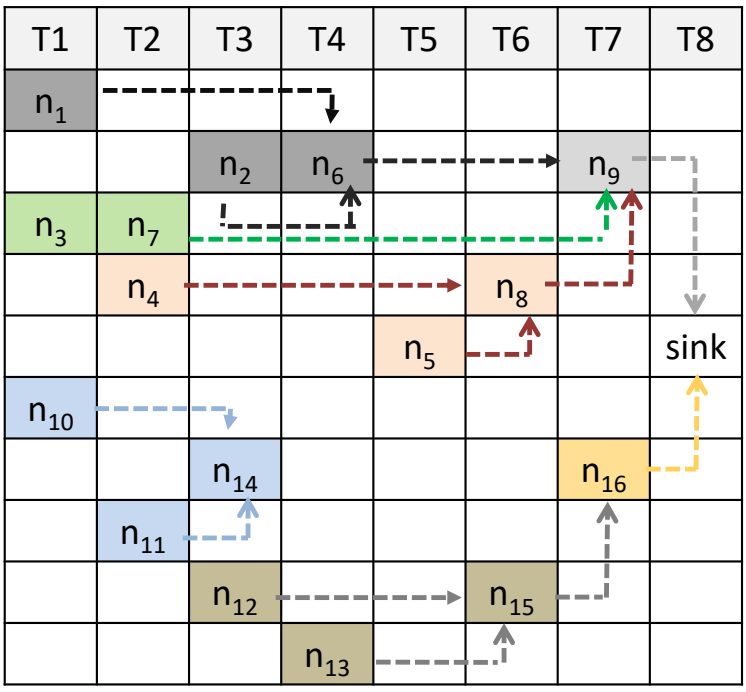


Figure 6: TDMA scheduling table

to omit redundant/repeated values [129]. If the primary parent does not receive any data from a child node, it raises a signal to its neighbors indicating that it did not hear any values from a specific child. In that case, if a backup parent receives this signal and overhears the child node data, it takes the responsibility of aggregating the value. Cascaded RideSharing can be thought of as token delegation in which the primary parent initially holds the token. The primary parent delegates the token in case it does not receive data from a specific child to the next backup parent. This process repeats among backup parents until one parent aggregates the value and releases the token. This token delegation process ensures that no sensor reading will be aggregated more than once.

In Cascaded RideSharing, there are three types of edges; primary and backup edges are the edges connecting each sensor node with its primary parent and backup parent in the adjacent track respectively, and side edges are those edges that connect parent nodes within the same track to ensure that each value is aggregated at most once. Note that a node can

be a primary parent node for one sensor and a backup parent for another sensor.

For error detection and correction purposes, each parent maintains a small bit vector L that has two bits for each child: r-bit (retransmitted bit) and e-bit (error bit) . As long as no error occurs in the primary edges, each primary parent receives the values of its children, aggregates them and sets the corresponding r-bit in the L vector for each corrected child to 1. If an error occurs in the primary edge, the primary parent sets the e-bit to 1 indicating a missing value from one of the children. Every parent attaches its bit vector to each message it sends. Other parents within range can overhear the L vector and decide whether to take any corrective action or not by examining the r-bits and e-bits. If the next backup parent in the correction order overhears the L vector with the r-bit set to 1, it takes no corrective action. In case of side edge errors, other backup parents take no corrective action.

Cascaded RideSharing ensures that (i) no sensor reading is included in the final aggregate more than once, and (ii) with high probability, each sensor reading will contribute to the final aggregate value. While the Cascaded RideSharing protocol achieves fault-tolerance of the individual sensor readings and their aggregate values, it has no privacy measures of guarantees. Simply encrypting the sensor readings would prohibit the ability of aggregating the sensor values and hence would break the Cascaded RideSharing protocol. Therefore, in the following section, a new privacy-preserving and fault-tolerance protocol for in-network aggregation of sensors data is presented to provide both privacy and high aggregate data precision with minimal performance overheads on the sensor nodes.

3.3 PRIVACY-PRESERVING AND FAULT-TOLERANCE PROTOCOL

In this section, the details of the proposed protocol for privacy-preserving fault-tolerance in-network aggregation are presented.

3.3.1 PROTOCOL OVERVIEW

At a high level, the privacy-preserving and fault-tolerant in-network aggregation protocol for CWSNs is built on top of the Cascaded RideSharing algorithm combined with the additively-homomorphic stream cipher described in Section 3.2. In the event that the readings of *all* sensor nodes are included in the final aggregate value, the algorithm goes as follows:

(i) each sensor n_i encrypts its value v_i as $c_i = v_i + g_i(k_i) \bmod M$

(ii) the resulting c_i values are aggregated using the Cascaded RideSharing protocol, which results in the sink receiving the value $C = \sum_i c_i \bmod M$

(iii) the sink then computes the aggregate key value $K = \sum_i g_i(k_i) \bmod M$

(iv) the sink extracts the final aggregate value $V = \sum_i v_i = C - K \bmod M$.

Unfortunately, the above algorithm only works in the rare case that *all* sensor nodes contribute readings to an aggregate computation. Most commonly, not all sensors' readings are included in the final aggregate because of either node- or link-level failures or simply because not every sensor will have a reading to contribute to every query. In this case, the sink node would compute an incorrect aggregate key K. If the sink attempts to decrypt the aggregate ciphertext using the wrong aggregate key, the resulting value will be a random element from the set $\{0, \ldots, M-1\}$. This random and unbounded error is due to the semantic security of the cipher, which ensures that a ciphertext reveals no information about the corresponding plaintext without the appropriate key.

To account for the above types of problems, the protocol is designed in a way that allows the sink node to *efficiently* determine which sensors contributed readings to the final aggregate and thus correctly compute the aggregate key that should be used to recover the true aggregate value from the ciphertext received. This is achieved by propagating state not only between nodes within the same track as done in Cascaded RideSharing, but also between nodes in adjacent tracks. Using this protocol, even duplicate-sensitive aggregates such as SUM and AVG are securely computed using the in-network aggregation process, while guaranteeing with high probability that every sensor reading contributes to the final aggregate at most once.

Algorithm 1: Aggregation and routing algorithm run by sensors within the network

input : PC, BC, SP, v

1 $A := 0$;
2 $PT := \bar{0}$;
3 $L.r := \bar{0}$;
4 $L.e := \bar{0}$;
5 **if** *v NOT NULL* **then** `// Aggregate own value`
6 $\quad A := A + v + g_{ID}(k_{ID}) \bmod M$;
7 $\quad PT[\text{ID}] := 1$;
8 **end**
9 $L :=$ rcv $(L(SP))$;
10 **foreach** *Child C in $PC \cup BC$* **do**
11 $\quad$ **if** *rcv(A_c,PT_c) from Child C* **then**
12 $\quad\quad$ **if** *$C \in PC$ OR ($C \in BC$ AND $L[C].e = 1$ AND $L[C].r = 0$)* **then**
$\quad\quad$ `// Aggregate the received values`
13 $\quad\quad\quad A := A + A_C \bmod M$;
14 $\quad\quad\quad PT := PT$ OR PT_c;
15 $\quad\quad\quad L[C].e := 1$;
16 $\quad\quad$ **end**
17 $\quad\quad$ **else** `// Propagate the error signal`
18 $\quad\quad\quad L[C].e := 1$;
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 **end**
22 Transmit(A,PT,L);

Algorithm 2: Final aggregation and decryption algorithm used by the data sink

input : PC

output: $FinalA$

$A := 0$;
$PT := \bar{0}$;
$K := 0$;
$FinalA := 0$;
foreach *Child* C *in* PC **do**
  if *rcv*(A_c,PT_c) *from Child* C **then**
    $A := A + A_C \bmod M$;
    $PT := PT$ OR PT_c;
  end
end
foreach *bit set to '1' in* PT **do**
  $K := K + g_i(k_i) \bmod M$;
end
$FinalA := A - K \bmod M$;

3.3.2 PROTOCOL DETAILS

Algorithm 1 contains pseudo-code describing the aggregation protocol as run by sensor nodes that help aggregate and route readings in the network, and optionally contribute their own readings to the aggregate being computed. This algorithm takes four inputs:

(a) a set of child nodes for which this node is the primary parent ("Primary" Children, or PC)

(b) a set of child nodes for which this node is a backup parent ("Backup" Children, or BC)

(c) the list of peer nodes in this track (set of peers, or SP)

(d) an optional sensor reading to include in the aggregation (value v)

In addition to maintaining the fault-tolerant L bit vector needed by the Cascaded RideSharing protocol (Section 3.2), Algorithm 1 also maintains a **Partaking vector**, called the PT vector, to keep track of nodes that have successfully contributed to the final aggregate. The PT vector is an n-bit vector, where n is the number of sensor nodes in the network. Section 3.5 shows that in practice, sensor nodes do not necessarily have to transmit the whole PT vector, but only a compressed vector to minimize bandwidth overhead.

Algorithm 2 contains pseudo-code describing the protocol run by the sink node requesting the aggregate. This algorithm takes only a single input: the set of children in track 1 of the graph (PC, similar to Algorithm 1). After the sink receives an encrypted value and a PT vector from each of its responsive children, it computes the sum of each such A value and the bitwise OR of every PT vector to compute both the final (encrypted) aggregate value and the final PT vector. From the final PT vector, the sink node can identify which sensor nodes successfully contributed with readings to the final aggregate. The sink then generates the corresponding keystreams for each contributing node and uses the aggregate key to recover the plaintext aggregate value (Section 3.2).

3.4 PROTOCOL PROPERTIES AND PROOFS OF CORRECTNESS

This section includes formal proofs that the privacy-preserving and fault-tolerant in-network aggregation protocol for CWSNs protocol detailed in Section 3.3 provides strong guarantees in terms of both sensor reading confidentiality and correctness of the final aggregation.

Theorem 1 (Confidentiality)**.** *During the execution of the protocol described by Algorithms 1 and 2, no sensor (except the sink) can learn the value of the readings reported by any other sensor, nor the value of any intermediate aggregate value.*

Theorem 1 follows directly from the semantic security of the cipher used by Algorithms 1 and 2 and the fact that each sensor node shares a unique symmetric key only with the sink.

Theorem 2 (Correctness)**.** *Under the assumption of "honest but curious" or "quiet infiltrators" attack nodes, the protocol described by Algorithms 1 and 2 includes each sensor reading*

at most one time during the aggregation process. Further, the sink node is able to correctly identify the sensors that contributed to this aggregate, generate the resulting aggregate key, and recover the correct result.

Proof. To prove this claim, the following must be asserted:

(i) each sensor reading is aggregated at most once,

(ii) the PT vector includes exactly the information needed to recover the aggregate key needed to decrypt the result.

Note that Algorithm 1 sets a bit in the PT vector if and only if the sensor reading for the corresponding node is included in the local aggregate. Also, the PT vector is always transmitted with the aggregate values. As such, proving assertion (i) is sufficient to prove the theorem and the proof will proceed by induction on the height of the track graph.

For the base case, consider a track graph consisting of three tracks: the sink (track 0) and two tracks of sensors (tracks 1 and 2). Assuming that track 1 has perfect connectivity to the sink, there is only a need to show that all readings from track 2 are aggregated by at most one node in track 1. Without loss of generality, assume that track 2 consists of a single sensor node n_i and there are 5 cases to be considered:

1. there are no link failures in the graph,
2. the link between n_i and its primary parent fails,
3. the link between n_i and its primary parent and some number of its backup,
4. there is a side-channel error in track 1 *prior to* the aggregation of n_i's reading,
5. there is a side-channel error in track 1 *after* the aggregation of n_i's value.

Case 1: If no links fail, the reading of sensor n_i will be heard by its primary parent. The primary parent will include this value in its local aggregate, set the r-bit for node n_i in its L vector, and transmit. Since the r-bit for node n_i is set, no backup parent in track 1 will take corrective action to include n_i's reading in its local aggregate.

Case 2: If the link between n_i and its primary parent fails, the primary parent will set the e-bit for n_i in its L vector and transmit this vector along its side-channel to the other nodes in track 1. The first backup parent of n_i will then incorporate n_i's reading into its local aggregate, set the r-bit for n_i in its L vector and transmit. No other backup parent in track 1 will take corrective action to include n_i's reading in its local aggregate since n_i's r-bit is now set in the L vector passed along track 1.

Case 3: If the links between n_i and its primary parent and between n_i and some number of its backup parents fail, n_i's primary parent will set the e-bit for n_i in its L vector to indicate an error. This L vector will propagate along the side-channel in track 1 until it reaches a backup parent that has overheard n_i's transmission. If no such backup parent exists, n_i's reading is lost. Otherwise, the first such backup parent incorporates n_i's reading into its local aggregate, sets the r-bit for n_i in its L vector and transmits. No other backup parent in track 1 will take corrective action to include n_i's reading in its local aggregate since n_i's r-bit is now set.

Case 4: If there is a side-channel error in track 1 *prior to* the incorporation n_i's reading in the aggregate, no entry will exist in the L vector for node n_i. As such, nodes optimistically assume that n_i's reading was already aggregated, and will not incorporate n_i's reading. This implies that n_i's reading will be absent from the final aggregate.

Case 5: If there is a side-channel error in track 1 *after* the incorporation of n_i's reading in the aggregate, this implies, as in the last case, that no entry will exist in the L vector for node n_i. As such, nodes optimistically (and correctly) assume that n_i's reading was already aggregated and will not again incorporate n_i's reading.

The above cases account for all possible link failure scenarios between tracks 2 and 1, and within track 1, and in all cases n_i's reading was included at most once. Thus, it is shown that Theorem 2 holds in the base case.

For the induction step, assume that Theorem 2 holds for all track graphs containing up to k tracks. Following is the prove that it also holds for all track graphs of up to $k+1$ tracks.

First, observe that an argument similar to that used in the base case shows that the reading reported by each sensor in track $k+1$ will be incorporated by at most one sensor in track k. Furthermore, the inductive hypothesis can be used to prove that the value reported by sensor in track k is incorporated at most once into the final aggregate. As such, the readings of sensors in track $k+1$ are incorporated at most once into the final aggregate, and Theorem 2 holds in all track graphs. □

Taken together, these theorems show that the protocol described by Algorithms 1 and 2 does indeed provide privacy-preserving and fault-tolerant in-network aggregation functionality for wireless sensor networks.

3.4.1 INTEGRITY CHECKING

Integrity checking of the data is another desirable property of CWSNs to defend against external attackers who could manipulate the data in transit. Even though integrity protection is not within the scope of this dissertation—given the assumed eavesdropping attack model—for completeness, this section will provide some ideas on how to add integrity to the proposed protocol.

One major problem with homomorphic cryptosystems is that they are *malleable* by design. An encryption algorithm is malleable if it is possible for an adversary, without knowing the secret key, to transform a ciphertext into another ciphertext that decrypts to a *related* plaintext.

Consider the cryptographic primitives in Section 3.2, where nodes n_1 and n_2 encrypt their i^{th} values v_1^i and v_2^i into the two ciphertexts c_1^i and c_2^i and the aggregate value $C^i = c_1^i + c_2^i$. An external attacker can jam the network while the aggregate C^i is being transmitted and then transmit an inflated (or deflated) aggregate value $C^{i'}$, by adding (or subtracting) some

constant to C^i *without* having the ability to decrypt the aggregate C^i. When the sink node attempts to decrypt the aggregate it receives $C^{i'}$ it will recover the modified value $V^{i'}$ rather than the true aggregate V^i. Unfortunately, such attacks are undetectable without adding extra cryptographic mechanisms to verify both the data integrity and the authenticity of the encrypted aggregate values across sensor nodes.

It is straightforward to extend this protocol with hop-by-hop integrity protections to guard against the injection or modification of data by malicious outsiders. In order to accomplish this, every sensor node can establish a shared secret key with all its parents in the adjacent track. Secure key distribution between nodes is a well-explored problem, and could be achieved using any secure and efficient key distribution scheme in the literature (e.g., [98, 53]). Using this shared key, the sensor node can compute cryptographic integrity code (e.g., a keyed HMAC [86]) over its aggregate value, and then transmit the aggregate along with its corresponding integrity code to its parents. The parent receiving these values can then verify both the integrity of the message, as well as authenticate that it was sent by one of its legitimate children. If the verification passes, the values are processed as usual. If the verification fails, the faulty value can be ignored by the receiving parent and, if necessary an alert can be raised indicating that tampering has been detected.

Note that in the case of malicious compromised aggregating nodes, integrity checking becomes a more complex problem as end-to-end integrity checking will be required. The solution for this problem is a proposed future work.

3.5 PERFORMANCE EVALUATION

In this section, the simulations conducted to evaluate the proposed algorithm are presented in details. The simulations and experiments evaluate both the communication and energy-consumption overheads associated with the proposed protocol. Different network densities, link failure rates and different contribution rates form the sensors are examined through three different sets of experiments. The results are presented along with a discussion of how the proposed protocol was able to achieve the desirable balance between the 3Ps.

3.5.1 SIMULATION SETUP

To understand the costs and benefits of the proposed protocol, the following four protocols were implemented by extending the WSN in-network aggregation simulator TiNA developed by the authors of [129]:

(i) a spanning-tree based aggregation protocol that provides neither fault-tolerance nor data confidentiality.

(ii) the Cascaded RideSharing protocol [64], which provides only fault tolerance.

(iii) the basic version of the proposed protocol described in Section 3.3, which provides both fault-tolerance and data confidentiality protection.

(iv) an enhanced version of the protocol protocol that applies compression (more specifically run-length encoding or RLE) to the PT vector to minimize data transmission overheads.

An *epoch* is defined as the execution of the protocol for a single instance of the query, that is, all nodes that have data to transmit do so and the in-network aggregation is carried out until the data reaches the sink. All protocols were compared against three main metrics:

- *Average relative RMS error*: The root mean square error in the final aggregate result, normalized to the correct result value. This metric measures how well each protocol handles node- and link-level failures.
- *Average energy consumed per node per epoch*: The average energy spent transmitting, listening for, and receiving data by each node for an epoch. This metric allows us to compare the power efficiency of each protocol.
- *Average message size transmitted per node per epoch*: The average amount of data transmitted by each node in the network during one run of the protocol. This metric assesses how much time each scheme utilizes the wireless channel.

In the simulations, a number of sensor nodes are distributed over a 320×320 ft^2 grid, with the data sink located closest to the center of this area. The radio range of each node is assumed to be 30 ft. Each simulation run consists of 30 epochs. All results presented are the averages over 10 simulation runs. The simulations execute a duplicate-sensitive SUM query. As in [129], the assumption is made that sensor nodes have the Mica2 mote power and

Parameter	Values
Total number of nodes	300, 400, 500, . . . ,1000, 1024
Link error rate	0.05, 0.10, . . . , 0.35
Number of primary+backup parents	at most 3
Participation level (% of nodes reporting values)	1.5%, 2.5%, 5%, . . . , 25%

Table 2: WSNs simulator parameters

bandwidth specifications [133]: data transmission consumes 65 mW, listening and reception consume 21 mW, and sensors in the sleeping mode consume no energy. Network bandwidth is assumed to be 38.4 $Kbps$ [64].

Link errors are assumed to occur independently of each other and are distributed uniformly throughout the network. Failures can happen independently in primary, backup, and side edges, while the links between track 1 and the data sink are assumed to be error free. The number of (primary and backup) parents for each node is set to be at most 3 and depends on the network (in particular, it is a function of network density). Table 2 lists some of the simulation parameters. Network activity is assumed to be much more costly than computation [76].

RC4 stream cipher was chosen as the pseudo-random keystream generator. The simplicity of the internal structure of RC4 (only additions, ANDs, and swaps) makes it very suitable for deployment on the Mica2 motes. Moreover, RC4's block size is only 8-bits with a memory footprint of only 428 bytes [122], so that it consumes around 10% of the total 4KB of available Mica2 memory. Yet, as previously mentioned, the protocol is not contingent to a particular stream cipher algorithm, any lightweight stream cipher or block cipher operating in stream mode can be used in place.

As an optimization and to conserve both energy and bandwidth, the Run Length Encoding (RLE) is applied to compress the PT vector at each sensor node. Using this performance optimization, each sensor node contributing a value sets the corresponding bit in the PT vector, aggregates any received PT vectors from its children into a single PT vector, compresses the new vector using RLE, and sends the compressed version. Note that, for any

sensor node to aggregate the received PT vectors, it does not necessarily need to de-compress all the vectors first before aggregating them. Using RLE, simple computations can be used to determine which bits in the compressed vector are set to '1' and correspondingly set those bits in the aggregate compressed PT vector.

With respect to the energy consumption of computations (including aggregation and RLE compression), the experiments rely on the fact that transmitting one bit over radio is at least three orders of magnitude more expensive in terms of energy consumption than executing a single instruction [76]. Through measurement, it was determined that the number of CPU cycles necessary to compress a PT vector of size 1024 bits is less than 20 cycles per bit on the average. Since this cost is greatly dominated by the cost of transmitting a single bit, the energy consumption due to compression was not considered in the simulations computations. The cost of the simple operations—such as additions, ANDs, ORs, and swaps—required by our stream cipher is similarly dominated by transmission cost, particularly in the event that sensors are pre-keyed and need not generate their keystream on the fly.

3.5.2 EXPERIMENTS AND RESULTS

All results presented in this section have a maximum error of less than 1% when computed with 95% confidence level, therefore error bars from the plots were omitted for better presentation.

3.5.2.1 EFFECT OF LINK ERROR RATE In this experiment, the number of sensor nodes was fixed in the network to 1024 nodes, statically deployed in a grid. The participation level is set at 100%; i.e., all sensor nodes participate by contributing readings to the aggregate computation. The link error rates are varied, where the link error rate is the probability by which a link would fail. That is, a link error rate of 16% indicates that this link with 16% chance will fail to transmit every time it attempts to, that is 16% of the packets will be lost.

Figure 7 shows the effectiveness of the Cascaded RideSharing scheme with respect to the error of the aggregated value for different link error rates, when compared with the spanning

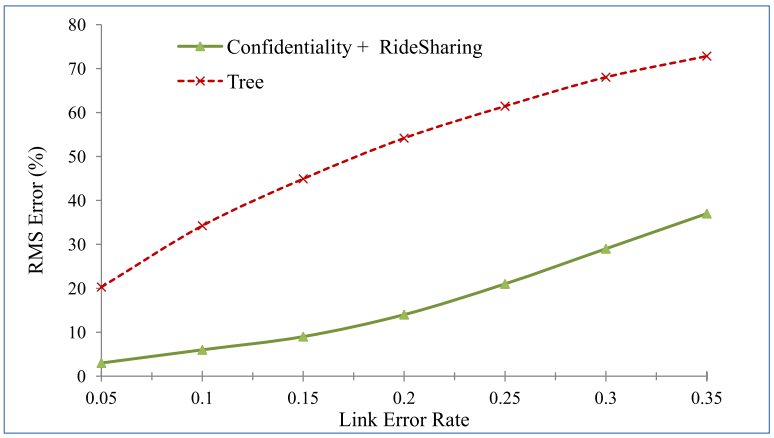


Figure 7: Average Relative Mean Square (RMS) error

tree scheme. As expected, link error rate is highly correlated with spanning tree error. For link error rate of 35%, there was an improvement in the RMS error of Cascaded RideSharing over the spanning tree scheme by 48.2%. This is because, with a spanning tree structure, a link failure causes the loss of a whole subtree of values.

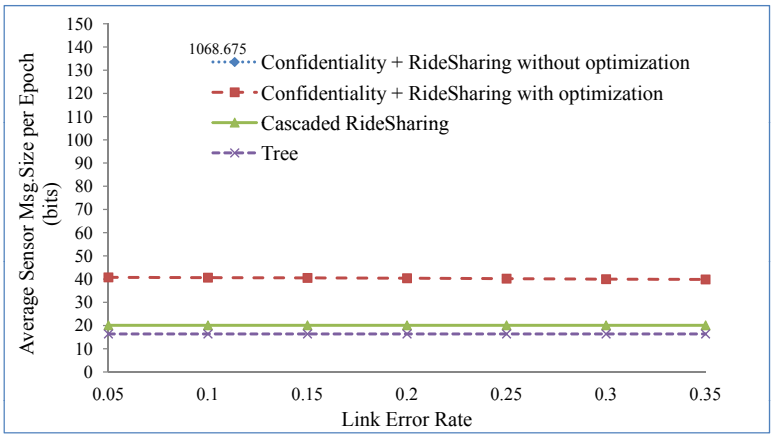


Figure 8: Average message size per node (per epoch)

Figures 8 and 9 illustrate the average message size and the average energy consumption overheads, respectively, for each scheme. The four schemes each show a stable overhead for different link error rates. Yet, the overhead differs significantly from one scheme to another.

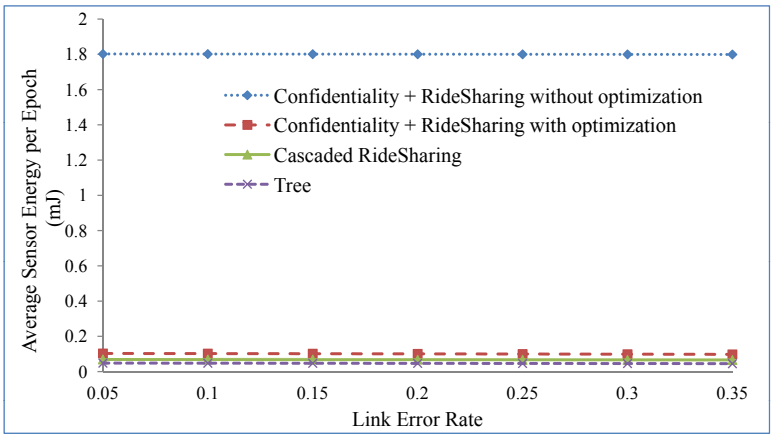


Figure 9: Average energy consumption per node (per epoch)

The naive version of our new scheme shows the maximum overhead[1] in both message size and the energy consumption. On the other hand, the version with compression optimization shows a significant improvement in the overhead over that without the compression optimization and is comparable to the schemes that do not provide fault-tolerance or confidentiality. A 96.2% reduction in the average message size and 94.5% reduction in energy consumption is achieved using the compression optimization. This is due to the fact that the PT vector starts from the higher network tracks with more "0" bits than "1" bits and then as the PT vectors gets aggregated towards the sink more "1" bits are introduced. Hence, the PT vectors are very compressible at both higher and lower level tracks.

3.5.2.2 EFFECT OF PARTICIPATION LEVEL A 100% participation rate is not a common case in collaborative WSNs, as often only a fraction of sensor nodes satisfies a query. Most queries involve a conditional clause, such as a WHERE condition. Only those sensors that satisfy the condition are expected to contribute to the query response. This experiment identifies the effect of the participation level on the overall system overhead. In particular, the link error is fixed to 0.25, with 1024 sensor nodes deployed in the grid. Only energy and message size are reported (RMS is relatively insensitive to participation level).

[1]It is 50x more than the baseline and thus not shown in the Figure

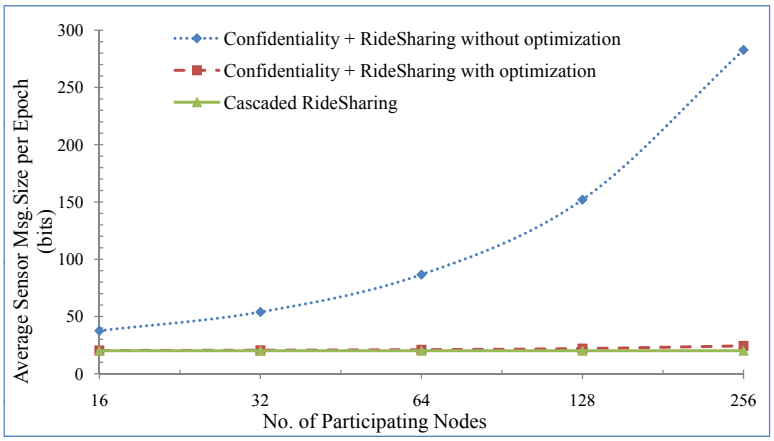


Figure 10: Average message size per node

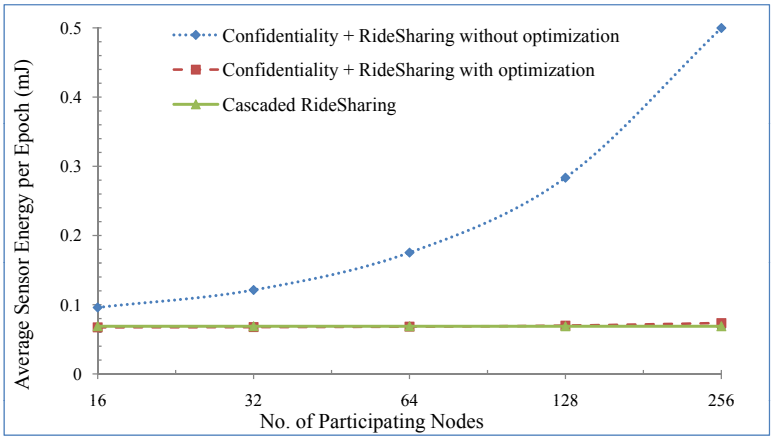


Figure 11: Average energy consumption per node

Figures 10 and 11, show a huge improvement in the overall system overhead when compared with the 100% node participation (cf.Figures 8 and 9).

The proposed protocol with compression optimization shows only an average of 7.1% and 3.6% increase in the average message size and average energy consumption, respectively, when compared with that of Cascaded RideSharing. Note that nodes that do not participate in the query result still need to send the L vectors to propagate the r- and e- bits among backup parents in the same track. This is necessary for the correctness of the fault-tolerance

scheme. Note also that with lower participation levels, the PT vector contains mostly "0" entries, hence the compression reaches excellent levels. For example, for 25% participation of the sensor nodes, the average message size per node decreases by 45% when compared with 100% participation for the same link error rate.

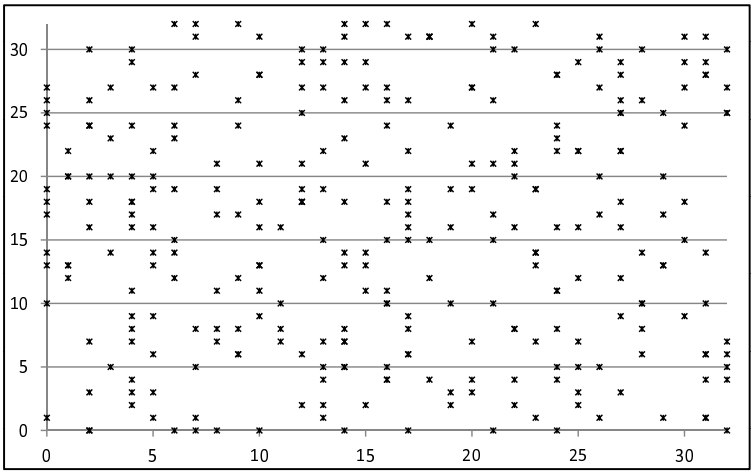


Figure 12: Example random deployment of 300 nodes in a 320 × 320 ft^2 grid

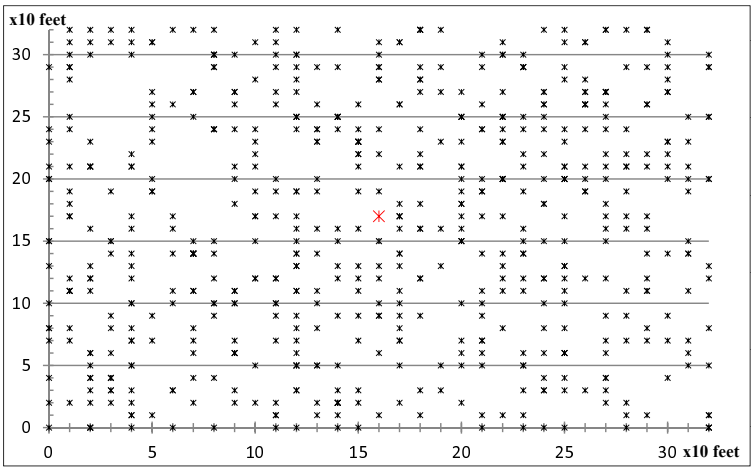


Figure 13: Example random deployment of 600 nodes in a 320 × 320 ft^2 grid

3.5.2.3 EFFECT OF NETWORK DENSITY In this experiment, the effect of the network density on the system performance is examined for the different schemes. The num-

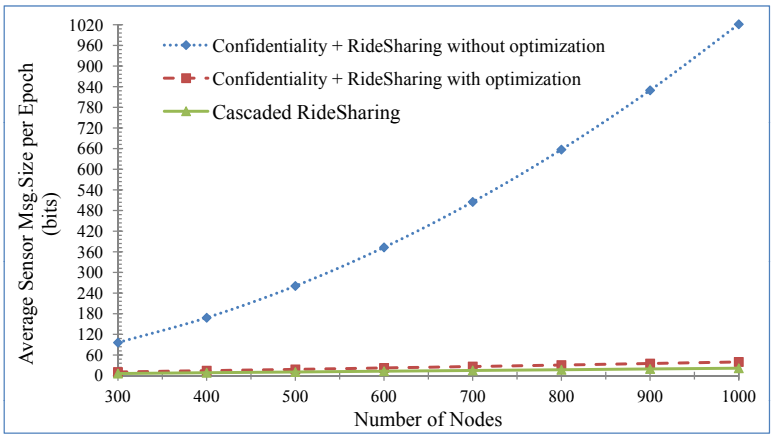


Figure 14: Avg. msg. size per node (with/without optimization)

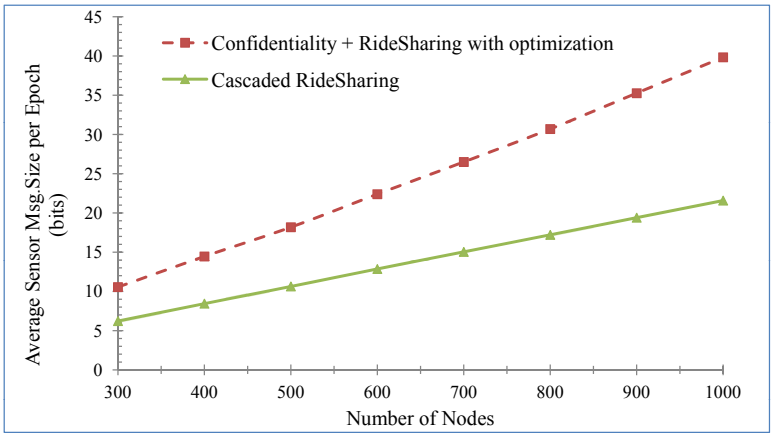


Figure 15: Avg. msg. size per node (with optimization)

ber of sensor nodes is varied from 300 to 1000 within the fixed 320 $\times$ 320 ft^2 grid. Using a uniform random distribution function, each sensor node is assigned a random (x, y) position in the grid. Figures 12 and 13 show sample random deployments of 300 and 600 nodes, respectively, in a 320$\times$320 ft^2 grid. Each simulation run then uses a different random network deployment, so the results represent the average over 10 different random deployments. The maximum number of parents per node is set to be at most three, the link error rate is fixed to 0.25, and sensors participation is 100%.

Figure 14 shows the average message size per node (per epoch) as the network density increases, while Figure 15 zooms in on the difference between the compression optimization and the Cascaded RideSharing scheme. Compression optimization reduces overhead: the average message size ranges from 10.5 to 39.8 bits per node (which represents from 60% to 80% overhead over the Cascaded RideSharing scheme), as the network density increases from 300 nodes to 1000 nodes. Figures 16 and 17 shows the average energy consumption overheads as the network density increases. Both figures illustrate the same metric, but Figure 16 shows the improvement of the compression optimization scheme over that without compression. The improvement in terms of average energy consumption per node is on average 90.2%.

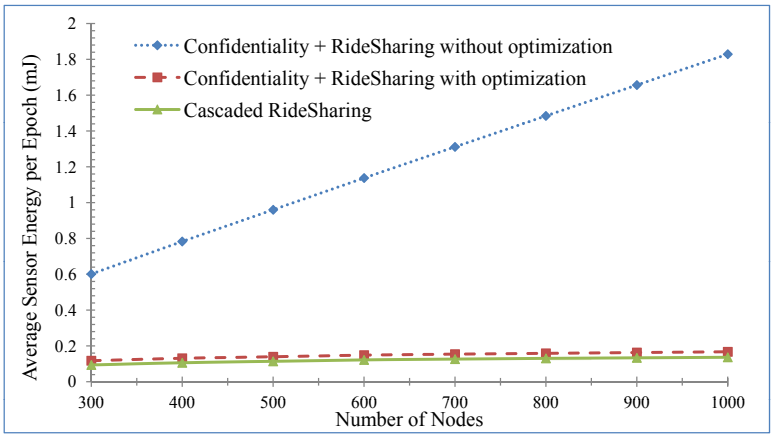


Figure 16: Avg. energy consumption per node (with/without optimization)

Figure 17 zooms in on the difference between the proposed scheme with compression optimization and the Cascaded RideSharing scheme. From the figure, the average energy consumption ranges from 0.11 to 0.16 mJ per node (representing an overhead that ranges from 22% to 25%), as the network density changes. It is worth mentioning that very dense deployments – as in the 1000 nodes case – are highly improbable deployments, therefore overheads of up to 25% and 80% of energy and message size, respectively, represent extremely unlikely situations.

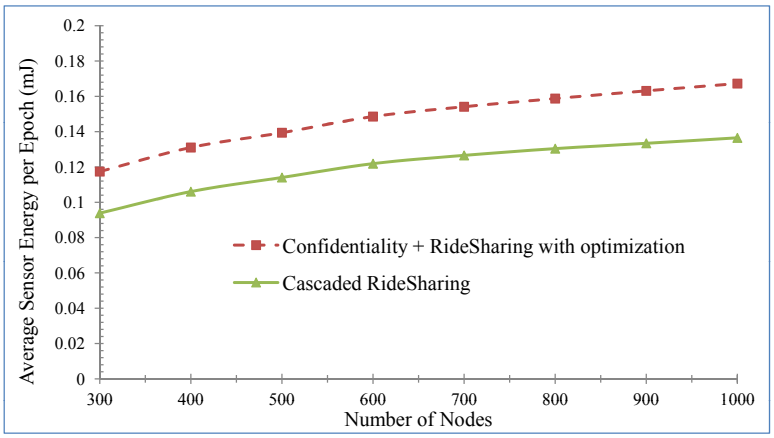


Figure 17: Avg. energy consumption per node (with optimization)

3.6 SUMMARY

The protocol presented in this chapter allows the aggregation of sensor readings while maintaining end-to-end confidentiality of both individual sensor readings and the aggregate results. The protocol makes use of a simple and efficient additive homomorphic cryptographic scheme and further offers robustness of the aggregation process via modifications to the Cascaded RideSharing fault tolerance scheme. The protocol guarantees that with high probability every sensor reading will contribute to the final aggregate through error detection and error correction techniques.

The proposed protocol successfully achieved a balance between *privacy*, *precision* and *performance*. Through experimental evaluations the aggregate data precision improved by 48.2% in the root mean square (RMS) error of the final aggregate result in comparison to the traditional spanning tree aggregate schemes. The experiments proved that both privacy and precision can be achieved in representative network configurations whole incurring an increase of 7.1% in the average message size and 3.6% in the average energy consumption. In the unlikely scenarios that 100% of the sensor nodes participate in an aggregate query, the average energy consumption overhead showed at most a 25% increase.

4.0 ENFORCING POLICY AND DATA CONSISTENCY OF CLOUD TRANSACTIONS

This chapter presents the second representative system application of this dissertation: enforcing policy and data consistency of cloud transactions [81, 82]. The proposed algorithms and solutions address the confluence of data, policy, and credential inconsistency problems that can emerge as transactional database systems are deployed to the weak underlying consistency models of the cloud.

In this system, the 3Ps requirements are defined as follows:

- **Privacy**: a transactional database system operating on cloud servers should ensure that at all times the access control policies enforced at multiple sites are in a consistent state during the execution of any single transaction. This requirement ensures that no unsafe access decisions are to be made at any point of time to protect sensitive resources and users privacy from being compromised.
- **Precision**: due to the weak consistency model of some cloud servers and the extensive use of data replication, transactional databases operating across database replicas must ensure the precision of the data retrieved from executing the queries. Data falling into an inconsistent state may lead to stale or inconsistent query results which affects the efficacy and credibility of query results. Accordingly, an agreement level across the transaction participants must be maintained.
- **Performance**: execution time of transactional database queries on the cloud servers should not be penalized due to the enforcement of policy and data consistencies across the multiple servers.

To achieve the desirable balance between the 3Ps requirements of this system while addressing the consistencies issues of policy-based transactional databases in the cloud, the following contributions are presented in this chapter:

- The first formalization of the concept of *trusted transactions* is presented. Trusted transactions are those transactions that do not violate credential or policy inconsistencies over the lifetime of the transaction. Then a presentation of a more general term, *safe transactions*, is introduced. Safe transactions identify transactions that conform to the ACID properties of distributed database systems *and* are trusted in terms of the validity of the policies evaluation.
- Different levels of policy consistency constraints are defined, as well as different enforcement approaches to guarantee the trustworthiness of transactions executing on cloud servers.
- A solution that involves an adaptation of the standard Two-Phase Commit (2PC) protocol to enforce trusted transactions, referred to as Two-Phase Validation Commit (2PVC) protocol is proposed. The protocol ensures that a transaction is *safe* and *trusted* as it ensures both policy and credential consistency along with ensuring data consistency.
- The performance of the proposed approaches is studied using both analytical and simulation based evaluations.

The rest of this chapter is organized as follows. Section 4.1 highlights the criticality of the inconsistency problems of policy-based transactional databases executing on the cloud infrastructures in the context of a motivational example. Section 4.2 introduces the system model and assumptions along with the formal definition of the problem. Section 4.3 describes the different levels of policy consistency constraints as well as the different enforcement approaches. In section 4.4, the proposed Two-Phase Validation Commit (2PVC) protocol is introduced in details and in Section 4.5 both the analytical and simulations based performance evaluations are presented. A trade-offs discussion of the 3Ps and the presented approaches is given in Section 4.6 to guide the decision makers to which approach to use in practice. Finally, Section 4.7 summarizes the work presented in this chapter.

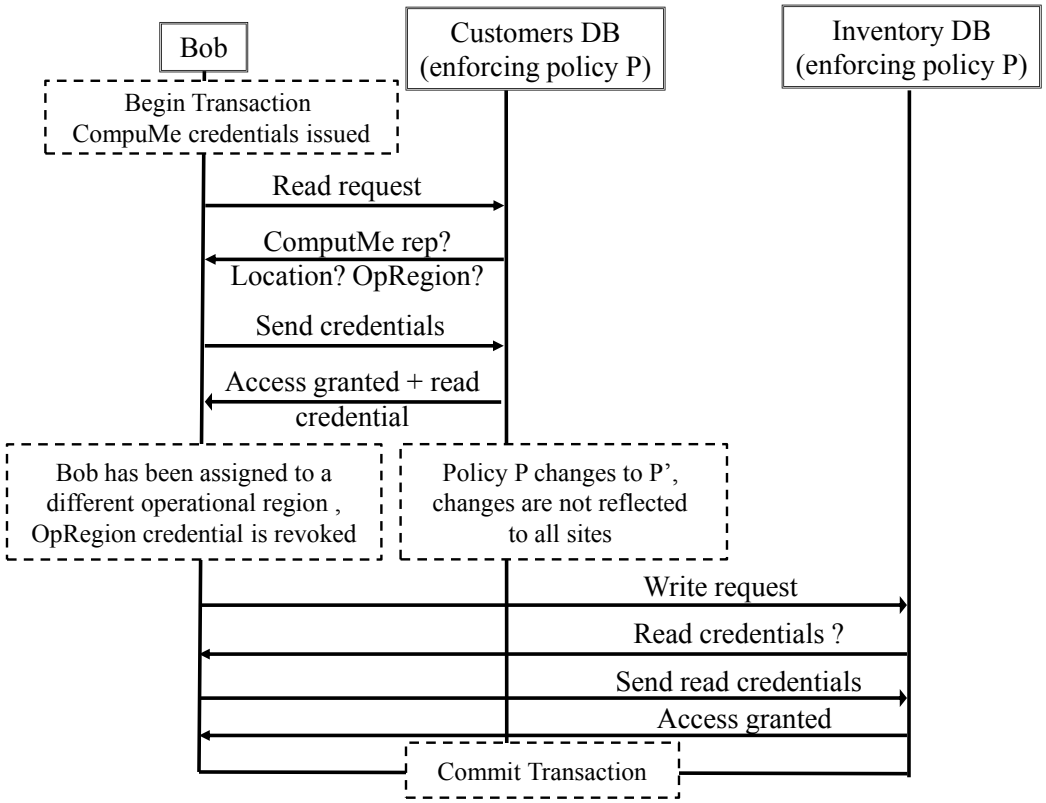


Figure 18: Motivating example: Sequence chart of Bob's interaction with the system

Figure 18 illustrates one case in which inconsistencies among policies and/or credentials of transactional databases executing on cloud infrastructures can cause unsafe authorizations to occur. In this scenario, Bob is working as a sales representative for CompuMe company that has a nationwide business for selling electronic equipment. Bob as a sales representative gets assigned by the company to conduct sales in a specific geographic region for a certain amount of time, then an assignment rotation takes place, and Bob's operational region is changed. As Bob is being assigned to one region, he is only allowed to access the customers' information and to report sales for his operational region and nowhere else.

The CompuMe company is hosting their backend databases and their web applications on cloud servers. The authorization policies that govern the read and write requests submitted to the databases through web transactions are also deployed on the cloud servers.

In this scenario, Bob is attempting to access a customer database that requires him to prove that he is a sales representative for his company (CompuMe), that he is currently assigned to sell within a particular geographical region, and that he is currently located within that region. Bob constructs such a proof of authorization, which is then verified by the customer database. The database then permits access, and returns Bob a credential indicating that he is permitted to read from the database.

Bob is then assigned to a different operational region, and the policy protecting resources within CompuMe is changed. However, since CompuMe makes use of an eventual consistency model for propagating policy changes, this new policy is not immediately propagated to all databases. When Bob attempts to access the inventory database, he is required to either satisfy (the original) policy, or present a previously-issued "read" credential indicating that the policy was satisfied. Bob presents his read credential, and is then granted access to the database. Note that Bob's second access was granted (i) using an old version of the access control policy and (ii) under the false pretense that Bob was still assigned to a valid operational region.

In general, many such problems can be encountered due to policy and/or credential inconsistencies. If any problems of policy consistency are not alleviated, the company or individual may suffer. The company may leak information about customers and face harsh penalties and loss of credibility. The individual may lose commission or other benefits for a sale.

4.2 SYSTEM ASSUMPTIONS AND PROBLEM DEFINITION

This section will present the system model for executing transactional databases on cloud servers. All system assumptions will be listed in details and the interaction between the different system components will be presented. The access control model used by the cloud servers in this system to govern the access to the data items in the transcational databases will also be presented in details. Next, the formalization of the new concepts, *trusted transactions* and *safe transactions*, that are introduced in this dissertation will be defined.

4.2.1 SYSTEM MODEL

The cloud infrastructure is assumed to consist of a set of $\mathcal{S}$ servers, where each server is responsible for hosting a subset D of all data items $\mathcal{D}$ belonging to a specific application domain ($D \subset \mathcal{D}$). In this system, the users interact by submitting queries or update requests encapsulated in ACID transactions. All the transactions submitted to the system are first forwarded to a *Transaction Manager* (TM). The main role of the TM is to distribute the submitted queries to the involved servers and coordinates their execution. Multiple TMs could be invoked as the system workload increases for load balancing, but the assumption is made that each transaction is handled by only one TM.

Transactions will be denoted as $T = q_1, q_2, \ldots, q_n$, where $q_i \in Q$ is a single query/update belonging to the set of all queries Q. The start time of each transaction is denoted by $\alpha(T)$, and the time at which the transaction finishes execution and is ready to commit is denoted by $\omega(T)$. Without loss of generality, the assumption is made that queries belonging to a transaction execute sequentially and without any restriction on the servers. That is, two queries of the same transaction may execute on the same server at different instances in time but not concurrently. These assumptions simplify the way the model and definitions are presented in this dissertation, but does not affect the correctness or the validity of the consistency definitions as presented in Section 4.3.

The set of all authorization policies will be denoted as $\mathcal{P}$. Each authorization policy $P : P \in \mathcal{P}$ enforced by a server s_i governing the access to the subset of data items D is defined as $P_{s_i}(D)$, where the policy P is a mapping such that $P : \mathcal{S} \times 2^{\mathcal{D}} \rightarrow 2^R \times A \times \mathbb{N}$. The value R indicates the set of inference rules used to define the authorization policy, A refers to the authorization policy administrator who is in charge of dictating an application's policy to the cloud servers, and $\mathbb{N}$ is the set of natural numbers and is used to identify the policy version v.

The set of all credentials will be referred to as $\mathcal{C}$. It is assumed that users' certified credentials are issued by an arbitrary number of *Certificate Authorities* (CAs) that exist in the system. It is further assumed that each CA offers an online method that allows any server to check the current status of a particular credential issued by the CA [105]. Given a

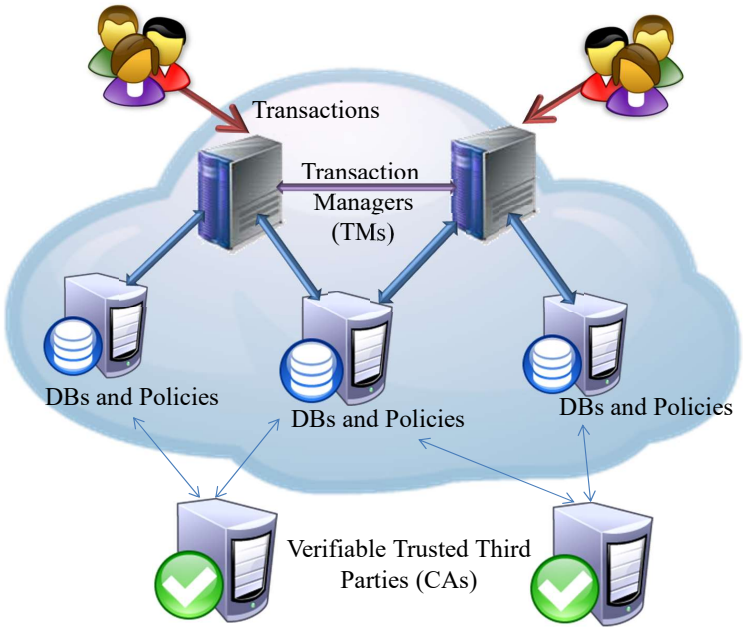


Figure 19: The interaction among the cloud system components

credential $c_k \in \mathcal{C}$, $\alpha(c_k)$ denotes the time at which the credential was issued by the CA, while $\omega(c_k)$ denotes the credential expiration time. Credentials can prematurely expire if they are revoked, and they can only be revoked by the issuing CA. Different cloud servers can also issue access credentials that act as *capabilities* allowing the user to continue submitting queries to other servers during the transaction lifetime (as was the case with Bob's read credential in Section 4.1). Servers can verify access credentials issued by each other.

Transactions are assumed not to fork out to other sub-transactions. This assumption is necessary to simplify the proof of correctness of the proposed schemes as presented later in Section 4.3. Transactions also do not externalize any data items to the users until commit time. Figure 19 illustrates the interaction among the different system components.

A proof of authorization evaluated at server s_i is defined as follows:

Definition 1. *(Proof of authorization)* $f_{s_i} = \langle q_i, s_i, P_{s_i}(m(q_i)), t_i, C \rangle$ *where* q_i *is a query defined over a set of read/write requests submitted to server* s_i*.* P_{s_i} *denotes the proofs of authorizations enforced by server* s_i *and belonging to the same administrative domain* A*. Function* m *is a mapping such that* $m : Q \rightarrow 2^D$*, that is,* m *identifies the set of data items that are being touched by query* q*. Time* t_i *is the time instance at which the proof of authorization is being evaluated, and* C *is a set of credentials presented by the querier to complete the proof of authorization such that* $C \subseteq \mathcal{C}$*.*

The set of all proofs of authorizations is denoted as $\mathcal{F}$, and the set TS contains all possible timestamps. The validity of each proof of authorization $f \in \mathcal{F}$ at time instance t is evaluated using the predicate $eval(f, t)$ such that $eval : \mathcal{F} \times TS \rightarrow \mathbb{B}$. The Boolean sign is true if the proof of authorization is valid. The validity of a proof of authorization is asserted in two cases:

- CASE I: Credentials are syntactically and semantically valid

  According to the definitions in [91], a credential c_k is syntactically valid if the following conditions hold:

  (i) it is formatted properly

  (ii) it has a valid digital signature

  (iii) the time $\alpha(c_k)$ has passed

  (iv) the time $\omega(c_k)$ has not yet passed.

  On the other hand, a credential c_k issued at time t_i is semantically valid at time t if an online method of verifying c_k 's status indicates that c_k was not revoked at time t' and $t_i \leq t' \leq t$.

- CASE II: The inference rules are satisfiable

  A policy is a set of inference rules that are encoded by policy makers to capture systems access control regulations. Given policy P, and user credentials C, if the inference rules of the policy can be satisfied using the user credentials, then the proof of authorization is said to be valid and the access is granted accordingly.

4.2.2 PROBLEM DEFINITION

This section presents the formalization of the new concepts introduced in this dissertation: **trusted transactions** and **safe transactions**. Trusted transactions are those transactions that do not violate credential or access control policies inconsistencies over the lifetime of the transaction. The more general term, *safe transactions*, is used to identify transactions that conform to the ACID properties of distributed database systems *and* are trusted in terms of the validity of the policies evaluation. Therefore, a safe transaction satisfies the correctness properties of proofs of authorizations, in which case is referred to as a *trusted transaction*, and also satisfies the data integrity constraints. Since data integrity and consistency has been extensively studied within the distributed database community (Chapter 6), this section will focus on defining the new concept of trusted transactions.

Since transactions are executed over time, the state information of the credentials and the policies enforced by different servers are subject to changes at any time instance, therefore it is important to introduce precise definitions for the different consistency levels that could be achieved within a transactions lifetime. These consistency models strengthen the trusted transaction definition by defining the environment in which policy versions are consistent relative to the rest of the system. For this reason, a transaction's *View* is defined in this dissertation in terms of the different proofs of authorizations evaluated during the lifetime of a particular transaction as follows:

Definition 2. *(View)*
A transaction's view V^T *is the set of proofs of authorizations observed during the lifetime of a transaction* $[\alpha(T), \omega(T)]$ *and defined as* $V^T = \{f_{s_i} \mid f_{s_i} = \langle q_i, s_i, P_{s_i}(m(q_i)), t_i, C\rangle \wedge q_i \in T\}$.

Following from Definition 2, a transaction's view is built incrementally as more proofs of authorizations are being evaluated by servers during the transaction execution.

In this dissertation, two increasingly powerful definitions of consistencies within transactions are presented.

Definition 3. *(View Consistency)*
A view $V^T = \{\langle q_i, s_i, P_{s_i}(m(q_i)), t_i, C\rangle, \ldots, \langle q_n, s_n, P_{s_n}(m(q_n)), t_n, C\rangle\}$ *is* view consistent*, or*

ϕ-consistent, if V^T satisfies a predicate ϕ-consistent that places constraints on the versioning of the policies such that ϕ-consistent$(V^T) \leftrightarrow \forall_{i,j} : ver(P_{s_i}) = ver(P_{s_j})$ for all policies belonging to the same administrator A, where function ver is defined as $ver : P \rightarrow \mathbb{N}$.

With a view consistency model, the policy versions should be internally consistent across all servers involved in the transaction. That is, a snapshot of the system is what is used to evaluate the decision of a trusted transaction with the servers agreeing among themselves. The view consistency model is weak in that the policy version agreed upon by the subset of servers within the transaction may not be the latest policy version v. It may be the case that a server outside of the S servers has a policy P that belongs to the same administrative domain A and with a version $v' > v$.

A more strict consistency model is the global consistency and is defined as follows:

Definition 4. *(Global Consistency)*
A view $V^T = \{\langle q_i, s_i, P_{s_i}(m(q_i)), t_i, C\rangle, \ldots, \langle q_n, s_n, P_{s_n}(m(q_n)), t_n, C\rangle\}$ is global consistent*, or ψ-consistent, if V^T satisfies a predicate ψ-consistent that places constraints on the versioning of the policies such that ψ-consistent$(V^T) \leftrightarrow \forall_i : ver(P_{s_i}) = ver(\mathcal{P})$ for all policies belonging to the same administrator A, and function ver follows the same aforementioned definition, while $ver(\mathcal{P})$ refers to the latest policy version.*

With a global consistency model, policies used to evaluate the proofs of authorizations during a transaction execution among S servers should match the latest policy version among the entire policy set $\mathcal{P}$, for all policies enforced by the same administrator A.

Both view consistency and global consistency models present two different degrees of the desirable precision property. Given the above definitions, a precise vocabulary for defining the conditions necessary for a transaction to be asserted as "trusted" can be presented as follows:

Definition 5. *(Trusted Transaction)*
Given a transaction $T = \{q_1, q_2, \ldots, q_n\}$ and its corresponding view V^T, T is trusted *iff $\forall_{f_{s_i} \in V^T} : eval(f_{s_i}, t)$, at some time instance $t : \alpha(T) \leq t \leq \omega(T) \wedge$ (ϕ-consistent$(V^T) \vee \psi$-consistent(V^T))*

Following from Definition 5, a safe transaction is a transaction that is both trusted and satisfies the data integrity constraints. A safe transaction is allowed to commit, while an unsafe transaction is forced to rollback. Hence, safe transactions provide guarantees in achieving both data precision and data privacy in cloud transactions.

4.3 TRUSTED TRANSACTIONS ENFORCEMENT APPROACHES

Following the definitions of safe and trusted transactions, this section will present several increasingly stringent approaches for enforcing trusted transactions. Each approach offers different guarantees during the course of executing transactions over cloud servers. This indicates that, like many other aspects of distributed systems, the choice of which approach to use is likely to be a strategic choice made independently of each application. Section 4.6 discusses in details the trade-offs to be considered when making the choice. The following sections will present the transaction enforcement approaches proposed from the most permissive and gradually all the way to the least permissive approach.

4.3.1 DEFERRED PROOFS OF AUTHORIZATION

Definition 6. *(Deferred Proofs of Authorization)*
Given a transaction T and its corresponding view V^T, T is trusted under the Deferred proofs of authorization approach, iff at commit time $\omega(T)$, $\forall_{f_{s_i} \in V^T}$: $eval(f_{s_i}, \omega(T)) \wedge$ (ϕ-consistent(V^T) $\vee$ ψ-consistent(V^T)).

Deferred proofs present an optimistic approach with weaker authorization guarantees, since the transaction queries are allowed to execute without being validated against the access policies. It is only at commit time when the proofs of authorizations are evaluated simultaneously and accordingly a decision is made whether the transaction is trusted or not. A Deferred proof of authorization has the choice of enforcing either view or global consistency

from Definitions 3 and 4 at commit time. The choice of which consistency level to enforce is a choice to be made by applications based on the required precision and privacy levels. By employing Deferred proofs of authorizations, transactions are most likely to execute faster but on the expense of risking a transaction to be forced to rollback after it has proceeded till the commit time if it violates the trusted transaction condition. These properties of Deferred proofs of authorizations show the interplay between the 3Ps in this system.

Figure 20 shows a scenario where three servers s_1, s_2, and s_3 are involved in the execution of a transaction. The horizontal lines define the transaction lifetime, and the dots represent the arrival time of a query to each server. The stars indicate the times at which each server validates a proof of authorization. The vertical dotted line represents an enforcement of either consistency or global consistency across the three servers. As shown in Figure 20a, the Deferred proofs of authorizations require only the proofs to be evaluated at the transaction commit time using either view or global consistency.

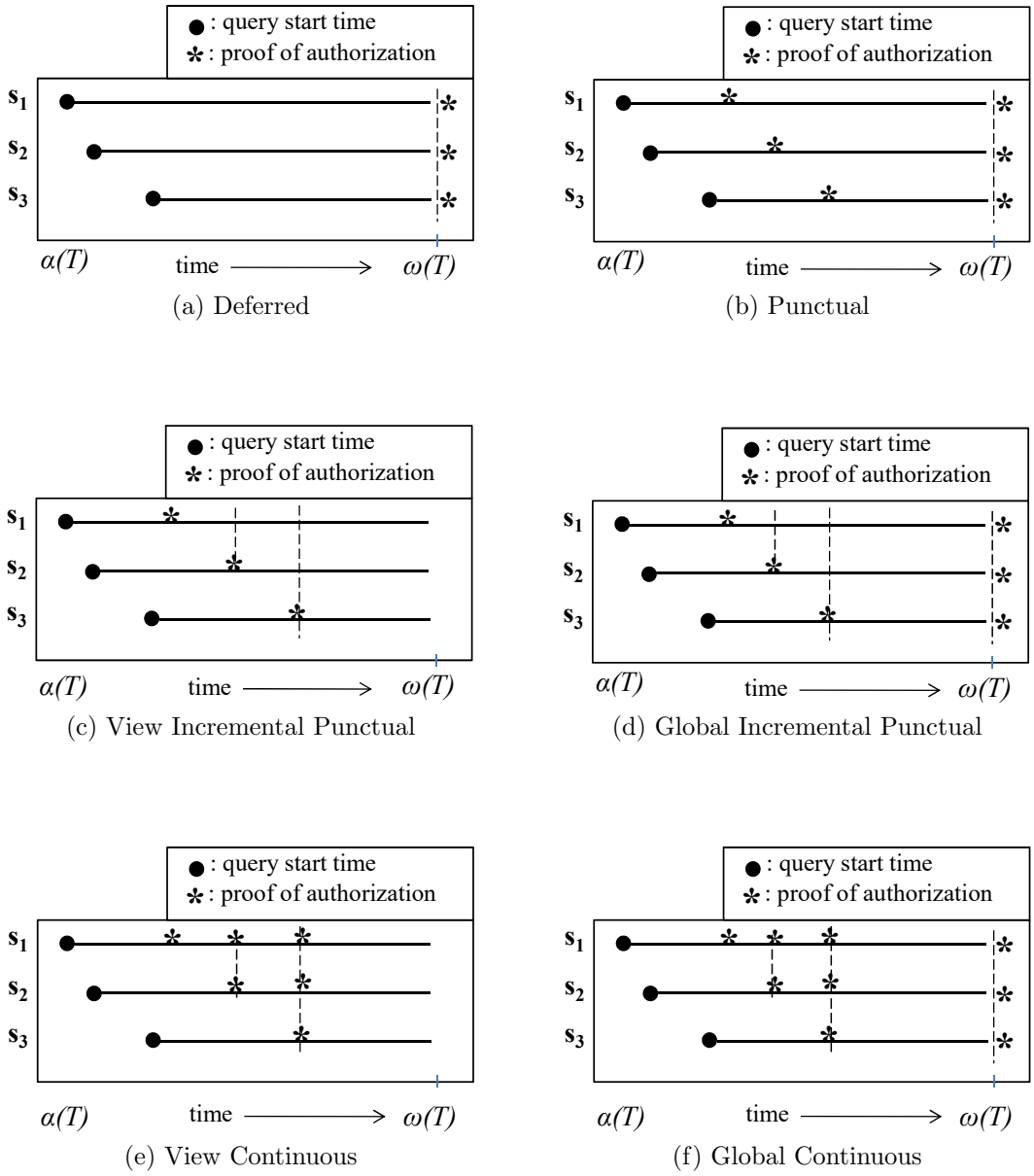
4.3.2 PUNCTUAL PROOFS OF AUTHORIZATION

Definition 7. *(Punctual Proofs of Authorization)*
Given a transaction T and its corresponding view V^T, T is trusted under the Punctual proofs of authorization approach, iff at any time instance t_i : $\alpha(T) \leq t_i \leq \omega(T)$ $\forall_{f_{s_i} \in V^T}$: $eval(f_{s_i}, t_i) \wedge eval(f_{s_i}, \omega(T)) \wedge$ (ϕ-consistent(V^T) $\vee$ ψ-consistent(V^T)).

Punctual proofs of authorizations present a more proactive approach in which the proofs of authorizations are evaluated instantaneously whenever a query is being handled by a server. A re-evaluation of all the proofs of authorizations at commit time is mandatory to ensure that throughout the window of execution of the transaction, policies were not updated in a way that would invalidate a previous proof, and/or credentials were not invalidated.

Using this approach, early decisions on whether a transaction should proceed or rollback could be made based on instantaneous evaluations of the proofs. Early detection of unsafe transactions can save the system from going into expensive undo operations.

As shown in Figure 20b, Punctual proofs of authorizations do not impose any restrictions on the freshness of the policies used by the servers during the transaction execution. It is



(a) Deferred (b) Punctual

(c) View Incremental Punctual (d) Global Incremental Punctual

(e) View Continuous (f) Global Continuous

Figure 20: Different variants of proofs of authorizations

only at commit time that either view of global consistencies are enforced. Hence, and due to the weak consistency paradigm on which cloud servers operate and during the transaction execution, each server might evaluate the proofs using a different policy version than the other participating servers. Consequently, servers might have false negative decisions and deny access to data, and also might have false positive decisions that could incorrectly allow

access to data. Therefore, the following sections will present two more restrictive approaches that enforce some degree of consistency among the participating servers each time a proof is evaluated.

4.3.3 INCREMENTAL PUNCTUAL PROOFS OF AUTHORIZATION

Before defining the Incremental Punctual proofs of authorization approach, a definition of the notion of *view instance*, which is a view snapshot at a specific time instance, is defined first.

Definition 8. *(View Instance)*
A view instance $V_{t_i}^T \subseteq V^T$ *is defined as* $V_{t_i}^T = \{f_{s_i} \mid f_{s_i} = \langle q_i, s_i, P_{s_i}^A(m(q_i)), t, C\rangle \in V^T \wedge t \leq t_i\}$, $\forall t, t_i : \alpha(T) \leq t \leq t_i \leq \omega(T)$.

Informally, a view instance $V_{t_i}^T$ is the subset of all proofs of authorizations evaluated by servers involved in transaction T up till the time instance t_i.

Definition 9. *(Incremental Punctual Proofs of Authorization)*
Given a transaction T *and its corresponding view* V^T, T *is trusted under the Incremental Punctual proofs of authorization approach, iff at any time instance* $t_i : \alpha(T) \leq t_i \leq \omega(T)$, $\forall_{f_{s_i} \in V_{t_i}^T}$ *: eval*$(f_{s_i}, t_i) \wedge$ *(*ϕ*-consistent*$(V_{t_i}^T) \vee$ ψ*-consistent*$(V_{t_i}^T)$*).*

Incremental Punctual proofs of authorizations develop a stronger conception of trusted transactions as a transaction is not allowed to proceed unless the desired level of the policy consistency at each participating server is achieved with all other servers. Figures 20c and 20d show that at each time instance whenever a proof of authorization is evaluated at a server, a vertical line is drawn to indicate that consistent policies among the servers is a requirement.

Without loss of generality, in Figure 20c, if the first server that starts executing the transaction has the latest policy version, in such case it is server s_1, all other servers (s_2, s_3) will be forced to have a consistent view with the first server before they can proceed with evaluating their proofs of authorization. In such a scenario, the chances of having a false positive or false negative authorization decisions is eliminated. On the other hand, if any of

the later servers have a newer policy version, that is, if any of the servers s_2 or s_3 receive a newer policy version than that used by s_1, the transaction is forced to abort. In the case of global consistency, the policies used by the servers need to be consistent with the freshest global policy, otherwise the transaction aborts.

For view consistency, in Figure 20c, at commit time, the view consistency condition will already be satisfied internally across all participating servers, therefore there is no need for another phase of consistency check. On the other hand, the global consistency condition necessitates another consistency check at commit time, as shown in Figure 20d. This extra check confirms that the policies used have not become stale – due to a policy update at some other server – during the window of execution between the last proofs of authorization and commit time.

4.3.4 CONTINUOUS PROOFS OF AUTHORIZATION

Continuous proofs of authorizations is the least permissive approach of all and is defined as follows:

Definition 10. *(Continuous Proofs of Authorization)*
A transaction T is trusted under the Continuous approach, iff $\forall_{1\leq i\leq n}\forall_{1\leq j\leq i}$ *:* $eval(f_{s_i}, t_i) \wedge eval(f_{s_j}, t_i) \wedge$ *(ϕ-consistent($V_{t_i}^T$)* $\vee$ *ψ-consistent($V_{t_i}^T$)) at any time instance* $t : \alpha(T) \leq t_i \leq \omega(T)$.

In Continuous proofs of authorizations, at every time instance when a proof is evaluated all previous proofs have to be re-evaluated if a newer version of the policy used is found at any of the participating servers. Similar to all other approaches, if any of the evaluations fail at any time instance, the entire transaction is forced to rollback. Figure 20e illustrates the Continuous proofs of authorizations given view consistency. Similar to the Incremental Punctual proofs, at commit time there is no necessity to force consistency among the servers again. However, in contrast with the Incremental Punctual proofs, if later executing servers are using newer policy versions than previous servers, all previous servers have to perform the following steps:

1. update their policy versions to be consistent with the newer ones,
2. re-evaluate their proofs of authorizations using the newer versions.

In the case of global consistency, as illustrated in Figure 20f, all servers will be forced to use the latest policy version at all times. Therefore, this variant is considered from all the previous approaches to be the most strict approach of all giving the best privacy and consistencies guarantees.

As mentioned earlier, the decision of which approach to adopt is to be handed to the policy administrators. As with any trade-off in most distributed systems, the stronger the security and precision guarantees given by an approach, the more the system has to pay in terms of implementation and messages exchange overheads. Further discussion of trade-offs issues and balancing the 3Ps will be presented in Section 4.6.

4.4 IMPLEMENTING SAFE TRANSACTIONS

As discussed in Section 4.2, a safe transaction is a transaction that is both trusted (i.e., satisfies the correctness properties of proofs of authorizations) and database correct (i.e., satisfies the data integrity constraints). This section describes an algorithm that enforces the execution of trusted transactions. Then, the algorithm is expanded to satisfy safe transactions, followed by a discussion of how these algorithms can be used to implement the various consistency approaches discussed in Section 4.3.

4.4.1 TWO-PHASE VALIDATION ALGORITHM

A common characteristic of most of the proposed approaches to achieve trusted transactions is the need for policy consistency validation at the end of a transaction. That is, in order for a trusted transaction to commit, its TM has to enforce either view or global consistency among the servers participating in the transaction. Toward this, a new algorithm called *Two-Phase Validation* (2PV) is proposed in this section.

As the name implies, 2PV operates in two phases: *collection* and *validation*. During the collection phase, the TM first sends a Prepare-to-Validate message to each participant. In response to this message, each participant (1) evaluates the proofs for each query of the transaction using the latest policies it has available and (2) sends a reply back to the TM containing the truth value (TRUE/FALSE) of those proofs along with the version number and policy identifier for each policy used in the proof evaluation. Further, each participant server keeps track of its reply (i.e., the state of each query) which include the id of the TM (TM_{id}) and the id of the transaction (T_{id}) to which the query belongs, and a set of policy versions used in the querys authorization (v_i, p_i).

Once the TM receives the replies from all the participants, it moves on to the validation phase. If all polices are consistent, then the protocol honors the truth value where any FALSE causes an ABORT decision and all TRUE causes a CONTINUE decision. In the case of inconsistent policies, the TM identifies the latest policy and sends an Update message to each out-of-date participant with a policy identifier and returns to the collection phase ignoring any FALSE replies. In this case, the participants perform the following steps:

1. update their policies,
2. re-evaluate the proofs,
3. send a new reply to the TM.

Algorithm 3 shows the process for the TM. In the case of view consistency (Definition 3), there will be at most two rounds of the collection phase. A participant may only be asked to re-evaluate a query using a newer policy by an Update message from the TM after one collection phase.

In the case of global consistency (Definition 4), the 2PV uses something akin to a master policies server to find the latest policy version. As such, the TM will retrieve this from some known master policies server in Step 2 and use it to compare against the version numbers of each participant in Step 3.

This master version may be retrieved only once or each time Step 3 is invoked. For the former case, the collection phase may only be executed twice as in the case of view consistency. In the latter case, if the TM is retrieving the latest version every round, global

Algorithm 3: Two-Phase Validation - 2PV(TM)

```
1  Send "Prepare-to-Validate" to all participants
2  Wait for all replies (TRUE/FALSE, and a set of policy versions for each
     unique policy)
3  Identify the largest version for all unique policies
4  if all participants utilize the largest version for each unique policy then
5  |  if any responded FALSE then
6  |  |  ABORT
7  |  end
8  |  else
9  |  |  CONTINUE
10 |  end
11 end
12 else
13 |  foreach participant with old versions of policies do
14 |  |  Send "Update" with the largest version number of each policy
15 |  |  Goto 2
16 |  end
17 end
```

consistency may execute the collection phase many times. This is the case if the policy is updated during the round.

The 2PV protocol executes at commit time and also during the execution of the transaction in the Continuous approach (Definition 10). In that case, when a query is requested, its TM will perform the following steps:

1. execute 2PV to validate authorizations of all queries up to this point,
2. upon CONTINUE being the result of 2PV, submit the next query to be executed at the appropriate server.

In the case of Incremental Punctual (Definition 9), a variant of the basic 2PV protocol is used during the transaction execution. In that approach, all participating servers need to execute their proofs of authorizations using a consistent policy version with the very first server that started executing the transaction. To achieve this, its TM keeps track of the policy version used by the first server, and checks it against the policy versions collected from the subsequent servers. If any of the subsequent servers have a newer policy version than that used by the first one, the TM decides an ABORT. In the global consistency case, the TM retrieves the latest policy version from the master policies server and checks that all servers are using that version, if not then again the decision is ABORT.

4.4.2 TWO-PHASE VALIDATE COMMIT ALGORITHM

Although 2PV provides trusted transactions, it does not satisfy the definition of a safe transaction as it does not validate the satisfaction of integrity constraints. Traditionally, integrity constraints in distributed systems are enforced by the *Two-Phase Commit* atomic protocol (2PC), which is a distributed agreement algorithm with two distinct phases: *voting phase* and *decision phase* [154]. There is a central TM that collects the decisions of each participant. In the voting phase, the participants involved in the transaction are polled for their vote on the commit. A YES vote from every participant is interpreted by the TM as a global agreement for a commit. On the other hand, a single NO vote from any participant induces a global rollback. In the decision phase, the TM notifies each participant with the voting decision and waits for an acknowledgment. Figure 21 illustrates the sequence of events of the basic atomic 2PC protocol.

In its basic format, 2PC cannot be used for satisfying safe transactions by combining integrity constraint validation and policy consistency validations because a response of YES (even if it were to suggest both data and policy consistency) would not indicate the version of the policy that the participant used to determine the authorization of the commit. There exists a situation where a participant says YES, when another participant has a fresher policy that would have contradicted the decision of the first participant. Therefore, in this dissertation, the proposed *Two-Phase Validation Commit* (2PVC) integrates both 2PV and

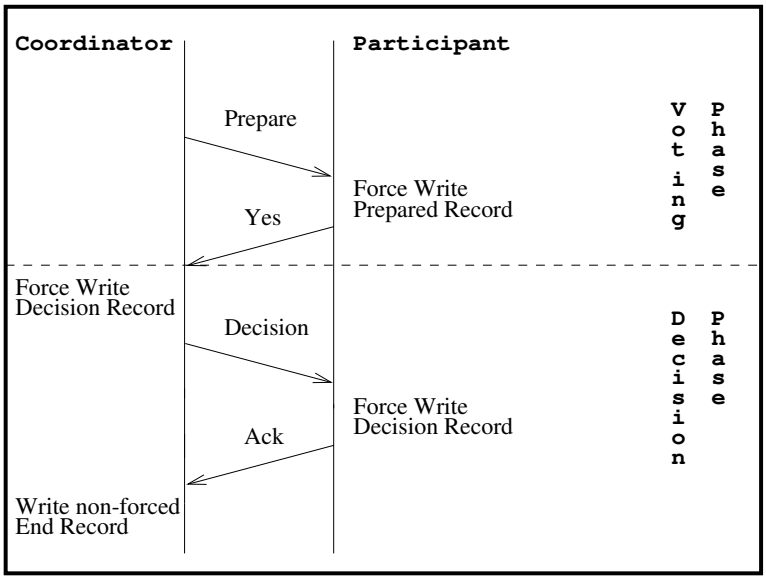


Figure 21: The basic two-phase commit protocol

2PC into a new. The 2PVC protocol is used to ensure both the data consistency and policy consistency of distributed transactions.

Specifically, 2PVC will evaluate the policies and authorizations within the first, voting phase. That is, when the TM sends out a Prepare-to-Commit message for a transaction, the participant server has three values to report:

1. the YES or NO reply for the satisfaction of integrity constraints as in 2PC,
2. the TRUE or FALSE reply for the satisfaction of the proofs of authorizations as in 2PV, and
3. the version number of the policies used to build the proofs (v_i, p_i) as in 2PV.

The process given in Algorithm 4 is for the TM under view consistency. It is similar to that of 2PV with the exception of handling the YES or NO reply for integrity constraint validation and having a decision of COMMIT rather than CONTINUE. The TM enforces the same behavior as 2PV in identifying policies inconsistencies and sending the Update messages. The same changes to 2PV can be made here to provide global consistency by consulting the master policies server for the latest policy version.

Algorithm 4: Two-Phase Validation Commit - 2PVC (TM)

```
1  Send "Prepare-to-Commit" to all participants
2  Wait for all replies (YES/NO, TRUE/FALSE, and a set of policy versions
     for each unique policy)
3  if any responded NO for integrity check then
4  |  ABORT
5  end
6  Identify the largest version for all unique policies
7  if all participants utilize the largest version for each unique policy then
8  |  if any responded FALSE then
9  |  |  ABORT
10 |  end
11 |  else
12 |  |  COMMIT
13 |  end
14 end
15 else
16 |  foreach participant with old policies do
17 |  |  Send "Update" with the largest version number of each policy
18 |  |  Wait for all replies
19 |  |  Goto 5
20 |  end
21 end
```

4.5 PERFORMANCE EVALUATIONS

This section presents both analytical and simulations based performance evaluations of the proposed consistency models as discussed in Sections 4.2 and 4.3.

4.5.1 ANALYTICAL COMPLEXITY

The cost of 2PC is typically measured in terms of log complexity (i.e., the number of times the protocol forcibly logs for recovery) and message complexity (i.e., the number of messages sent). We add another metric, namely the number of proof evaluations. These metrics are given with respect to the number of participants involved with the decision, n, the number of queries, u, and the number of voting rounds, r. The log complexity of 2PVC is no different than normal 2PC, which has a log complexity of $2n + 1$ [45].

	Deferred		Punctual		Incremental	
	View	Global	View	Global	View	Global
messages	$2n + 4n$	$2n + 2nr + r$	$2n + 4n$	$2n + 2nr + r$	$4n$	$4n + u$
proofs	$2u - 1$	ur	$u + 2u - 1$	$u + ur$	u	u

	Continuous	
	View	Global
messages	$u(u + 1) + 4n$	$u(u + 1) + u + 2n + 2nr + r$
proofs	$\frac{u(u+1)}{2}$	$\frac{u(u+1)}{2} + ur$

Table 3: The complexities of the various proofs of authorizations schemes

Table 3 shows the complexity – in terms of the maximum number of messages and proofs – for each proof of authorization scheme for both view and global consistency. Generally, 2PVC requires $2n + 2nr$ messages, where there are $2n$ messages for the voting phase (which may be repeated r times) and $2n$ messages for the decision phase. With view consistency, the number of voting rounds r is at most 2 (one extra round when compared to 2PC). For the case of Deferred and Punctual under view consistency, only the 2PVC is used. As such, they both require $2n + 4n$ messages in the worst case (r is 2). For Incremental and Continuous, however, consistency is maintained throughout the transaction which fixes r to 1. In the case of Incremental Punctual, the 2PVC is invoked without validations for a total of $4n$ messages. Continuous proofs of authorization perform 2PV at each query, which requires communication with potentially one extra server for each subsequent query executed. As such, the number of messages for 2PV is given by $2\sum_{i=1}^{u} i$, which is equal to $u(u + 1)$. By

adding the $4n$ messages of the 2PVC without validations, the total becomes $u(u+1)+4n$.

In terms of proofs, the general 2PVC will evaluate u queries each round for ur overall proof evaluations. For the case of a view consistency, 2PVC will evaluate the first round by validating all u queries. For the second round, at least one query will not have to be re-evaluated as it is the one providing the latest policy. Therefore, at most $2u-1$ proofs are required. Since Deferred uses only the 2PVC, it requires those $2u-1$ proofs. Because Punctual also evaluates proofs during the transaction, it adds an extra u proofs totally $u+2u-1$. Incremental Punctual, as mentioned previously, maintains the policy consistency as the transaction executes and, as such, does not require 2PVC with validations. In this case, it simply evaluates the u proofs during the transaction execution. Since Continuous proofs of authorization perform 2PV at each query, it will require an extra proof for each subsequent query executed. Similar to the number of messages, the number of proofs during the transaction execution are given by $\sum_{i=1}^{u} i$ which is equal to $\frac{u(u+1)}{2}$ proofs. It does not require 2PVC with validations at commit time since running 2PV at the final query does the equivalent work.

In the case of global consistency, r is not bounded. The Deferred and Punctual schemes now require the general $2n+2nr$ messages plus r messages to receive the latest policy version per round. The proofs for both must account for extra rounds during 2PVC giving ur and $u+ur$ proofs, respectively. Both Incremental Punctual for global and view consistencies have the same complexity. The global version has one difference as it retrieves the global version number every query for an extra u messages on top of the $4n$ for 2PC giving a total of $4n+u$. The number of proof evaluations is the same since 2PVC with validations are not required. Finally, Continuous uses the general 2PVC along with u messages to get the master version number for the u invocations of 2PV and r messages to get the master version number for 2PVC. The total messages become $u(u+1)+u+2n+2nr+r$. Since the 2PVC validations must now be performed, an extra ur proofs are added in the global consistency giving a total of $\frac{u(u+1)}{2}+ur$.

4.5.2 SIMULATIONS

4.5.2.1 ENVIRONMENT AND SETUP Each of the proofs of authorizations approaches along with the different consistency levels (view and global) as described in Section 4.3 were implemented using the Java programming language. Although the approaches were implemented in their entirety, the underlying database and policy enforcement systems were simulated with parameters chosen according to Table 4. To understand the performance implications of the different approaches, the following parameters were varied:

(i) protocol used
(ii) level of desired consistency
(iii) the frequency of master policy updates
(iv) transaction length
(v) number of servers available

The experimentation framework consists of three main components: a randomized transaction generator, a master policy server that controls the propagation of policy updates, and an array of transaction processing servers. The first server that starts executing a transaction is considered the Transaction Manager (TM) and holds the responsibility of coordinating the rest of the servers participating in the execution of the transaction.

The experiments were run within a research lab consisting of 38 Apple Mac Mini computers. These machines were running OS X 10.6.8 and had 1.83 GHz Intel Core Duo processors coupled with 2GB of RAM. The experimental code was compiled and run using Java SE 1.6.0 update 33. All machines were connected to a gigabit Ethernet LAN switched with a Cisco Catalyst 4006 over Cisco WS-X4448-GB-RJ45 switch boards. Round trip times between machines on this network were approximately 0.35 ms. All WAN experiments were also conducted within this testbed by artificially delaying packet transmission by an additional 75 ms.

For each simulation and each possible combination of parameters, 1000 transactions were run to gather average statistics on transaction processing delays induced by the particular protocol and system parameter choices. The randomized transaction generator module generated transactions that were randomly composed of database reads and writes with equal

probability. To simulate policy updates at different servers, the master policy server picks a random participating server to receive the updates.

The following assumptions were made to simplify the experiments and help limit the influence of other factors on transaction execution time:

1. only one single master policy server exists and has to be consulted for the latest policy version
2. all proofs of authorizations evaluate to *true* (i.e., only authorized individuals attempted to run transactions in the system)
3. no data integrity violations were encountered during transactions execution

Parameter	Value(s)
Number of policies updates	once during operations, once per participant join, or once at commit time
Disk read latency	1-3 ms
Disk write latency	12-20 ms
Authorization check delay	1-3 ms
Data integrity constraint verification	1-3 ms
Transaction size	Short: 8-15 operations, Medium: 16-30 operations, or Long: 31-50 operations

Table 4: Cloud database transactions simulation parameters

Therefore, transactions would only abort due to policy inconsistency. This means that Deferred, Punctual, and Continuous proofs will always produce successful commits, whether or not a policy change is detected.

The following are the measurements taken during the simulations:

1. The average execution time of the *shortest* successfully committed transactions (denoted t_s); it occurs when there are no policy changes.
2. The average execution time of the *longest* successfully committed transactions (denoted t_f); it occurs when policy changes force re-evaluations of the proofs of authorizations or multiple rounds of 2PV are invoked (e.g., in Continuous proofs).

Essentially, t_f captures the cost of recovering from a conflict. In the case of Continuous proofs, the worst case is when a policy change is detected each time a new server joins the execution of a transaction. The average transaction execution time to terminate (abort or commit) for Deferred, Punctual, and Continuous proofs can be computed using the following equation, where P_u represents the probability of a policy update:

$$t = t_s(1 - P_u) + t_f P_u \tag{4.1}$$

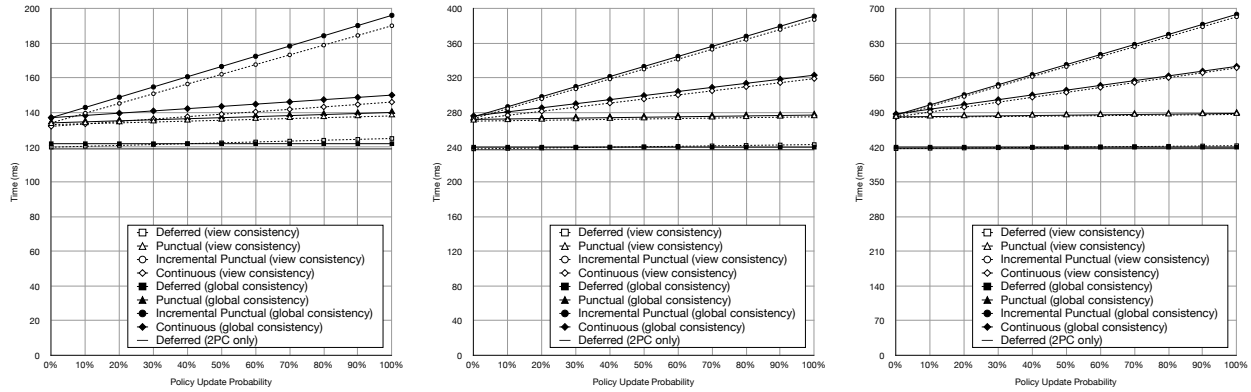
That is, a 0% chance of policy update means that no recovery from policy changes will be required during the transaction, whereas a 100% chance of policy update means that each transaction will need to respond to a policy change.

As opposed to the other proofs of authorization, in Incremental Punctual proofs, if a policy change is detected during the execution of a transaction, the transaction will abort regardless it is using view or global consistency. Therefore, to compute the average execution time, we assume that each aborted transaction is re-executed once to successful commit, with all servers using consistent policies. This assumption approximates the cost for rolling back the aborted transactions. The following equation is used to compute the average transaction execution time:

$$t = (t_f + t_s)P_u + t_s(1 - P_u) \tag{4.2}$$

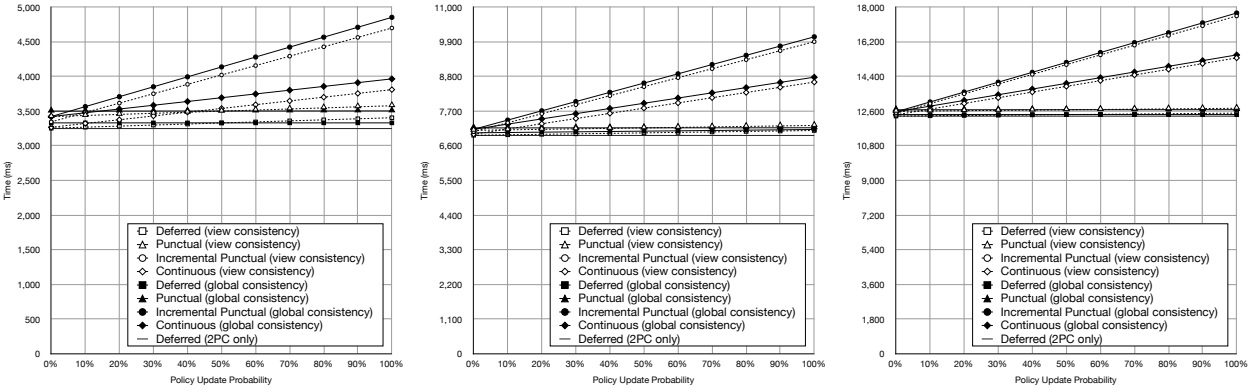
where t_f denotes the measured average time of the quickest aborted transactions among the simulation runs, and t_s denote the average time of the successfully committed transactions.

4.5.2.2 SIMULATION RESULTS Using Equations. 4.1 and 4.2, Figures 22 and 23 were plotted to show the simulation results for both the LAN arrangement and the simulated WAN, respectively. Each figure shows the execution time of the committed transaction (y-axis) as the probability of the policy update changes (x-axis). The figures contrast between the four different approaches for proofs of authorizations each with the two validation modes, namely, view and global consistency. The figures show different transactions length: (a) short transactions involve 8–15 operations running on up to 5 servers, (b) medium transactions



(a) Short Transactions (8–15 operations) (b) Medium Transactions (16–30 operations) (c) Long Transactions (31–50 operations)

Figure 22: Simulation results for LAN experiments



(a) Short Transactions (8–15 operations) (b) Medium Transactions (16–30 operations) (c) Long Transactions (31–50 operations)

Figure 23: Simulation results for WAN experiments

involve 16–30 operations running on up to 15 servers, and (c) long transactions involve 31–50 operations running on up to 25 servers.

For each case, and as a baseline, the transaction execution time is measured when transactions execute without any proof of authorizations and are terminated using the basic 2PC (shown in figures as a solid line referring to Deferred 2PC only). In all cases, the average transaction execution time of Deferred proofs with 2PVC was effectively the same as the

baseline indicating that 2PVC has negligible overhead over the basic 2PC.

The relative performance of the different proofs of authorizations is consistent throughout the different experiments. From the figures, it can be concluded that the Deferred proofs have the best performance of all, as the transaction operations are allowed to proceed without interruption until commit time. Of course, proofs of authorizations failing at commit time will force the transaction to go into a potentially expensive rollback. That will not be the case with the other schemes, as the proofs are evaluated earlier during the execution of the transactions and the rollback process of aborted transactions involves fewer operations.

Punctual proofs come next in terms of performance. The minor difference between Punctual and Deferred proofs is because Punctual proofs incur the cost for the local authorization checks each of which is in the range of 3-5 ms. Both Deferred and Punctual proofs are on average insensitive to the probability of policy updates (as realized from the graph slope). This is due to the fact that both schemes only enforce consistency at commit time.

Incremental Punctual proofs show the worst performance of all schemes and are the most sensitive to the probability of policy updates. This is due to the fact that Incremental Punctual proofs using either view or global consistency have the risk of aborting and re-executing each time a policy update is encountered. As the policy update probability increases, the performance of Incremental Punctual is severely penalized.

Continuous proofs show better performance than the Incremental Punctual approach, but are worse than the Deferred and Punctual approaches. Just as with Incremental Punctual, the performance of Continuous proofs suffers as the probability of policy update increases, as with each policy update all previously evaluated proofs will go through a re-evaluation phase using the 2PV protocol.

A final observation is that in most cases, global consistency proofs are slightly slower than view consistency. This extra latency comes from the additional communication round between TM and the master policy server to retrieve the latest policy version. Global consistency proofs were faster than view consistency ones in the few cases when the latest policy happens to match the policy used by all participating servers and as a result all servers skip the re-evaluation step of 2PVC.

4.6 3PS TRADE-OFF DISCUSSION

This section presents a discussion of the strengths and weaknesses of each of the proposed protocol variants as presented in Section 4.3 relative to the 3Ps. Since choosing a scheme for data and policy consistency enforcement is a strategic decision that has to consider many trade-offs, the impact of application level requirements on this decision is also discussed in details.

4.6.1 ABSOLUTE COMPARISON

Table 5 represents a comparison between each of the presented eight approaches for consistency enforcement. In the table, all approaches are assessed relative to one another using the metrics described above and the rankings used (high, moderate, low and poor) are not absolute; rather each approach is assessed comparatively.

From the table, it is made clear that scoring "high" in all the metrics is not possible, indicating a trade-off between performance and the other metrics. By design, all the approaches are highly precise, as all participants internally agree on the policy version to be used during the transaction, although the accuracy of this policy can vary. In the general case, view consistency is less accurate than global consistency, since the view of the servers executing the transaction might not account for recent updates to the policy in use. Precision and accuracy together define the *security* and *privacy guarantees* of the system. *Deferred* proofs are less accurate and thus less secure than the other approaches, as policies remain unchecked until the end of a transaction, but have the best performance overall. The *Punctual* approach has similar performance to Deferred, but is slightly more accurate than Deferred, since authorizations are at least checked locally throughout the transaction. *Incremental Punctual* has the worst performance, especially when frequent policy updates occur, but is more accurate and secure than the previous approaches. Finally, *Continuous* has moderate to low performance which is the penalty that must be paid for the high accuracy afforded by this approach.

	Deferred		Punctual		Incremental		Continuous	
	View	Global	View	Global	View	Global	View	Global
Performance (infrequent updates)	■	■	■	■	⊟	⊟	⊞	⊞
Performance (frequent updates)	■	■	■	■	□	□	⊟	⊟
Precision	■	■	■	■	■	■	■	■
Accuracy	□	⊟	□	⊟	⊞	■	⊞	■
	■ – high		⊞ – moderate		⊟ – low		□ – poor	

Table 5: Contrasting evaluation of the various proofs of authorizations

4.6.2 IMPACT OF APPLICATION REQUIREMENTS

Choosing a consistency enforcement scheme is not something that can be done in isolation, as application requirements may limit the schemes available for use within a particular application. Typically applications consider three orthogonal axes of requirements: *code complexity* (which is directly related to trusted computing base size), *transaction mix* (i.e., write-only, read/write with internal reads, and read/write with materialized reads), and *policy/credential update frequency*.

Following, two cloud-based applications that are representative of larger classes of interesting applications are investigated to show how these requirements can impact the choice of the consistency enforcement scheme.

Application: Event Scheduling

Consider an Event Monitoring Service (EMS) used by a multi-campus university to track events within the university and to allow staff, faculty members, and student organizations to make online event registrations. The university is using a cloud infrastructure to host the various EMS databases and execute the different transactions. Users have varying access

privileges that are governed by authorization policies and credentials issued by a university-wide credentialing system. In this system, read requests must be externalized to users during the transaction execution so that intermediate decisions can be made. Furthermore, the university system in general has infrequent policy and credentials updates, and requires lower code complexity to minimize code verification overheads.

Recommendation: In this case, the use of Punctual proofs makes the most sense. This approach has low code complexity, performs faster, and is suitable for systems with infrequent updates. Read externalization is also permissible, as policies are checked prior to each operation during the execution of the transaction.

Application: Sales Database

This example is derived from the traveling salesperson example in Section 4.1. According to company requirements, a customer's data should only be read by the representatives in the operations region of that customer, while any other data should not be materialized until commit time. The company also experiences very frequent policy and credential updates, as representatives are frequently assigned to different operational regions. The company considers security to be very important as to avoid incorrect authorization decisions that might leak customer information. Finally, the company has enough resources to manage complex code, but still requires reasonable execution latency.

Recommendation: This company should use the Continuous Global consistency scheme for the highest accuracy to avoid any information leakage at runtime, or Continuous View for slightly lower accuracy. This provides a good balance between accuracy and performance yet at the cost of higher code complexity.

4.7 SUMMARY

This chapter identified several consistency problems that can arise during cloud-hosted transaction processing using weak consistency models, particularly if policy-based authorization systems are used to enforce access controls. Therefore, this dissertation introduced a novel set of light-weight proof enforcement and consistency models—namely, Deferred, Punctual,

Incremental, and Continuous proofs, with view or global consistency models—that can enforce increasingly strong protections with minimal runtime overheads.

Simulated workloads were used to experimentally evaluate implementations of the proposed consistency models relative to three core metrics: transaction processing performance, accuracy (i.e., global vs. view consistency and recency of policies used), and precision (level of agreement among transaction participants). The simulations showed that high performance comes at a cost: Deferred and Punctual proofs had minimal overheads, but failed to detect certain types of consistency problems. On the other hand, high accuracy models (i.e., Incremental and Continuous) required higher code complexity to implement correctly, and had only moderate performance when compared to the lower accuracy schemes.

In the end, the choice of which consistency model to use is application dependent. For instance, if the transactions need to materialize reads, then either Incremental or Continuous proofs need to be used to avoid information leakage at runtime. On the other hand, if low code complexity is required – e.g. to facilitate formal protocol verification– the Deferred or Punctual models should be preferred, which necessitates relaxing consistency requirements. To better explore the differences between these approaches, a detailed trade-off analysis of the proposed schemes was presented to show that most application needs can be met using one of the eight protocol variants proposed in this chapter.

5.0 DATA STREAM MANAGEMENT SYSTEMS

This chapter introduces the third and last representative distributed system applications of this dissertation: *privacy-aware, shared-operator networks* for Data Stream Management Systems (DSMSs). In DSMSs, continuous queries (CQs) are registered by users in the system to process streaming data that originates from different data sources. To improve performance and optimize the usage of system's resources, the query optimizer typically groups CQs that share common sub-expressions (i.e., computational steps that evaluate the same value), and generates a shared-operator network for the execution of these grouped queries. While sharing the execution of queries has proven to enhance the performance, enforcing access control for the privacy preservation of the data streams becomes an issue. The solution presented in this chapter introduces a novel technique to satisfy access control policies at run-time in shared-operator networks for DSMSs reinforcing both the *need to share* and *need to protect* design models. The proposed algorithm provides a cost effective way for shared-operator networks to enforce access control in an undisruptive efficient manner eliminating the need of switching between different operator networks in the case of intermittent changes in the access control.

In this system, the 3Ps requirements are defined as follows:

- **Privacy**: DSMSs should protect the privacy of the data providers from the unauthorized access of their data streams and the query results produced on these streams by enforcing the access control policies that the data providers specify.
- **Precision**: changes made to the shared-operator networks to accommodate the possible changes in the access control permissions should not affect the correctness of query results (i.e, no tuples should be lost or dropped while these changes are being made).

- **Performance**: applying access control on shared-operators networks should be a cost effective process that sustains the cost benefits of shared operators.

To achieve the desirable 3Ps properties while enforcing access control policies over shared-operator networks in DSMSs, the following contributions are presented in this chapter:

- A new set of low overhead streaming operators, referred to as "Privacy Switches" (PrSs), are proposed. These switches dynamically allow or deny the flow of data streams in the shared-operator networks based upon the current state of the access control permissions. They can temporarily halt the execution of certain branches in the shared-operator networks denying access to some query results in the case of access loss. Later, once access is re-gained, these branches can resume operation again on the newly arriving data.
- A PrSs placement algorithm is designed to identify the most cost-effective placement positions of the PrSs in any given shared-operators network. The strategic placement of the switches ensures seamless execution of the CQs and reduces the overheads in the networks while honoring all possible changes in access permissions.
- The performance of the proposed privacy-aware shared-operator networks is studied through simulations of randomly generated query networks.

The rest of the chapter is organized as follows. Section 5.1 introduces the concepts of shared-operator networks through an example of healthcare data streams. Section 5.2 presents the system model, assumptions and preliminaries. Section 5.3 describes the proposed approach and algorithms. Section 5.4 presents the experimental evaluation and finally Section 5.5 summarizes the work presented in this chapter.

5.1 MOTIVATING EXAMPLE

Along with the development of the internet of things (IoT) technology, wearable devices based on IoT are being actively developed for and used in various applications. In particular, the technology development of wearable devices that can continuously monitor human activity and bio-signals using sensors has played a major role in the development of the

smart healthcare industry [84], [42]. For example, with the wide spread use of wearable devices having various bio sensors, it is possible to easily measure and monitor diverse health data such as blood glucose levels, blood pressure, oxygen saturation, heart rate, and body temperature of individuals [41], [112]. This, in turn, makes it possible to provide an alarm service, notifying in the risk of health issues based on individual activities of daily living.

The development of the smart healthcare industry brings forth a need for collecting largescale personal health data in order to leverage the knowledge obtained through analyzing such data for improving smart healthcare services. For example, Apple Health [3], Google Fit [6], and Samsung S-Health [10] are capable of providing vast amounts of health-related stream data using smartphones and wearable devices such as a smartwatch and smartband. Different health care applications can continuously query these stream data to detect health issues, notify the users or provide general healthcare guidelines.

Even though a large collection of health data is a valuable asset to the smart healthcare field, serious concerns of data privacy are being raised. That is, indiscriminate collection of personal health data can cause significant privacy issues; sensitive information of individual users can be deduced by tracking and analyzing health data. Hence, most users do not agree to their health data being collected for the purposes of data analysis and utilization. This presents a major obstacle for the development of smart healthcare services.

The following example demonstrates three different data streams that can be used in smart healthcare applications (Listing 5.1). The first stream, Stream1, contains health and location data being produced by a fitness watch linked to a phone. The second data stream, Stream2, contains location and travel data generated by a smart phone, while the third data stream, Stream3, captures users spending time on electronic devices and is generated by multiple electronic devices associated with a particular user.

```
Stream1: streamid, location, heartRate, timestamp
Stream2: streamid, location, speed
Stream3: streamid, location, screentime, category
```

Listing 5.1: Data streams generated by smart devices

Different data consumers might be interested in querying these stream data. For example, a certified health monitoring application might issue a CQ to determine if a user is having health issues while driving by checking whether they are in their car while their heart rate is elevated. Another CQ might be issued by another application that performs research studies on the effect of sitting still and using screen time for extended hours on the health conditions of users.

Standard DSMSs use queries optimizer to find efficient execution plans for different CQs. From the query execution plans, an operator network can be constructed for each query in SQL. DSMSs can execute query networks separately. However, there might be some common filtering conditions that are found among the queries. For example, the operators may be performing the same restriction on the input data stream or performing the same join between two data streams, which are referred to as common sub-expressions. To reduce the overall processing and memory costs incurred by individually executing queries, queries optimizer can group together multiple queries that have common sub-expressions into single interleaved execution plans.

Furthermore, DSMSs can enforce access control policies to control who can query and process certain data streams. Assume that there are multiple smart healthcare services that have access to the input data streams or only certain users who hold the right set of credentials are allowed to view and query the data streams. For example, an access control policy of the form: $p_1 = ((healthmonitoringApp \wedge certifiedApp) \vee doctor \vee firstResponder)$ could be used to govern the access of both Stream1 and Stream2 indicating that the users of the certified health monitoring application or users holding doctors or first responders' credentials are allowed to execute queries on those streams.

Another policy: $p_2 = ((healthresearchApp \wedge certifiedApp) \vee researcher)$ could be one of the policies used to govern access to Stream3 where only certified health research applications or users holding health researchers' credentials are allowed to query this stream.

To enforce the access control policies, given that access control policies can change or users' credentials can also change, DSMSs need to identify who can have access to the query results. If any of the users lose their privileges to access any of the input data streams and hence the query results, the straight forward solution would be to apply post-filtering to

the query results. That is, enable or disable different users from accessing the final query results based on their access privileges. Yet, consider the case were all users lose their access to one of the queries, in this case there are two options: 1) post-filter the results and pay the penalty of having all the query operators execute to retain the shared-operators network functioning disruptively, or 2) halt the shared-operators network and execute each query independently until access is granted again to all queries then execute the shared-operator network again. Note that, in this option, not only do multiple different copies of the operator networks need to be maintained and their states synchronized each time there is a change in access to a query, but also multiple copies of the input streams have to be provided for each network in the system. It is obvious that options 1 and 2 are both inefficient ways of handling intermittent loss of access to the queries.

Therefore, this chapter introduces a simple and novel solution to achieve privacy in DSMSs by enforcing access control policies over shared-operator networks. The solution guarantees that the precision of the query results is always maintained and that the systems performance is never compromised.

5.2 PRELIMINARIES AND SYSTEM MODEL

This section will start by introducing the DSMSs system model as well as the system assumptions. Then later, operators and shared-operator networks are formally defined. Finally, the access control model and the preliminaries used by the proposed system will be introduced.

5.2.1 SYSTEM MODEL

The system consists of four main classes (entities): *data providers*, *data users/consumers*, *DSMS*, and *authorization servers* (Figure 24).

- *Data Providers*: generate and distribute data streams to DSMSs. They have the choice of creating and updating the access control policies for their generated data streams to grant or deny access to different data users.

- *Data Consumers*: can be individuals or applications who submit CQs to the DSMSs or register in any set of predefined queries in the system. Data consumers also submit the necessary credentials required for the proofs of authorizations to grant them access to their registered queries.
- *DSMSs*: handle all the incoming data streams and submitted queries. They execute query optimizers to generate shared-operator networks for the interleaved execution of queries and schedule the operators execution. They are in charge of enforcing the access control policies on the data streams and the corresponding query results (i.e., only those applications or users that are allowed access to certain data streams can view the corresponding query results executing over those streams).
- *Authorization Server*: trusted servers that can be separate entities or an integral part of the DSMSs. They can identify the access control permissions of the different data streams and the submitted queries of the data consumers, based upon the policies defined by the data providers and/or DSMSs. Accordingly, they can run their own proofs of authorizations and validate users' credentials to grant or deny access of users to certain data streams and queries. Authorization Servers inject special *Security Punctuations* into the input data streams (explained in details in Section 5.2.3) that are used to identify the access control permissions on a per user basis.

5.2.2 OPERATORS AND OPERATOR NETWORKS

This section introduces the main defining elements of data streams, CQs, and operator networks that will be used throughout the remainder of this chapter.

- DSMSs are capable of processing long running set of CQs $q_1, q_2, \ldots, q_n$ executing over data streams in the system.
- $\mathcal{S}$ denotes the set of all input data streams in the system, while a continuous data stream $s \in \mathcal{S}$ is a potentially unbounded sequence of tuples that arrive over time.
- Tuples in a data stream are of the form $t = [sid, tid, A]$, where sid is the stream identifier, tid is the tuple identifier, and A is the set of attribute values of the tuple.
- Queries are comprised of a set of query operators $o_1, o_2, \ldots, o_n$.

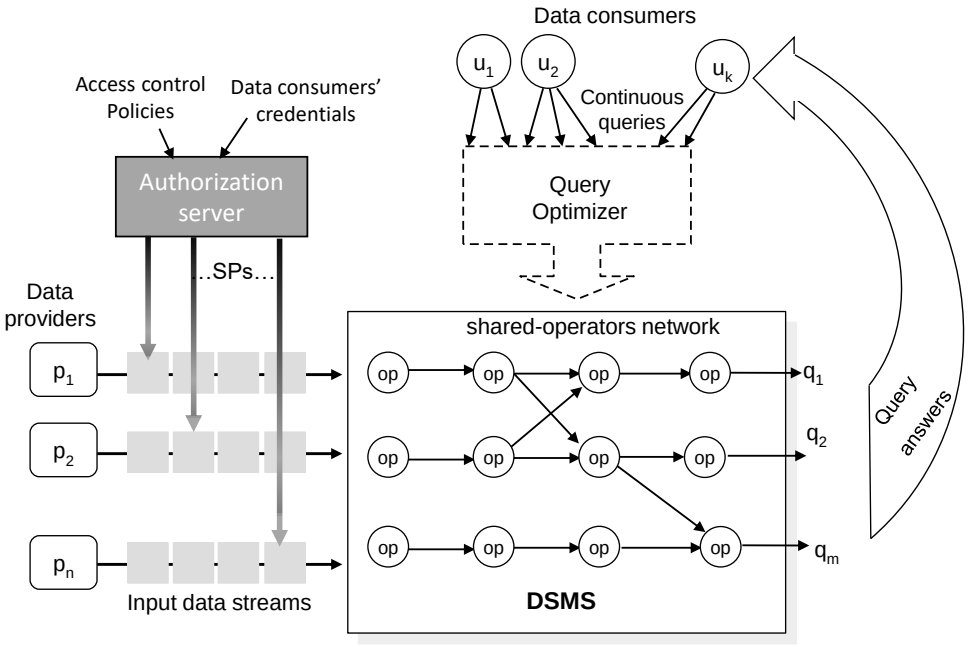


Figure 24: DSMS system model

Query operators are defined as follows:

Definition 11. *A query operator o_i is presented as $operator_{predicate}$ and take the form of relational operators with predicate that can be a conjunction of* `SELECT` *operator σ over the same data stream or a conjunction of* `JOIN` *operator $\bowtie$ between different data streams. It can also be a* `PROJECT` *operator Π over a data stream to filter out certain attributes to the query output, and finally it can be an* `AGGREGATE` *operator with predicate over a single attribute of a data stream to perform some algebraic aggregate function, such as:* `MAX`, `COUNT`, `SUM`, *etc.*

For example, the following are valid query operators

$expr_1$: $o_1 = \sigma_{s_2.speed<30}$

$expr_2$: $o_2 = \bowtie_{s_1.location=s_2.location\ \&\ s_1.timestamp>6{:}00am\ \&\ s_1.timestamp<7{:}00pm}$

Listings 5.2 and 5.3, illustrate the two example CQs that can be defined over the data streams introduced in the motivating example in Section 5.1.

```
SELECT s1.streamid, s1.heartRate
FROM Stream1 as s1 [RANGE 5 minutes, SLIDE 1 minute],
     Stream2 as s2 [RANGE 5 minutes, SLIDE 1 minute]
WHERE s1.location = s2.location
AND s1.timestamp > 6:00am
AND s1.timestamp < 7:00pm
AND s2.speed < 30;
```

Listing 5.2: Q1

```
SELECT s1.streamid, s1.heartRate
FROM Stream1 as s1 [RANGE 10 minutes, SLIDE 1 minute],
     Stream2 as s2 [RANGE 10 minutes, SLIDE 1 minute],
     Stream3 as s3 [RANGE 10 minutes, SLIDE 1 minute]
WHERE s1.location = s2.location
AND s2.location = s3.location
AND s1.timestamp > 8:00am
AND s1.timestamp < 7:00pm
AND s2.speed < 0
AND s3.screentime > 5h;
```

Listing 5.3: Q2

When the CQs are processed, only those data tuples that satisfy the query operator predicates are produced as results. As mentioned earlier in Chapter 2, there are two types of operators in CQs: *stateless* and *stateful* operators. A stateless operator, such as `SELECT`, produces an output tuple based solely on the current input tuple. Conversely, a stateful operator, such as `JOIN` or `AGGREGATE`, needs to refer to values from previous input tuples.

Due to the fact that input streams are infinite, DSMSs use either *tumbling* or *sliding* windows, to define the scope of the executions of operators and limit their state. Sliding windows allow the output to be continuously computed based on the most recent "portion" of the stream data. A sliding window is specified through a length (or range) l, and a slide

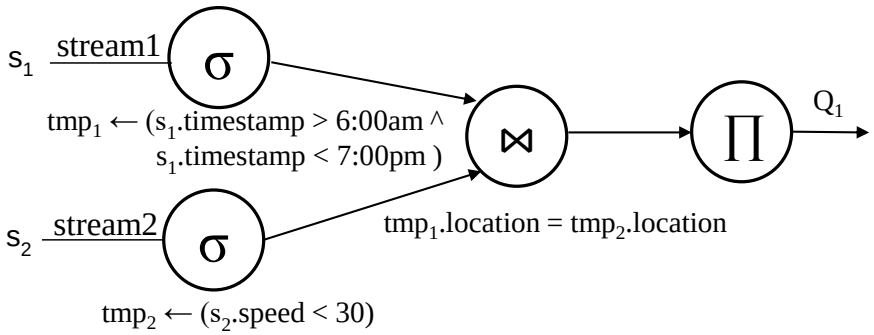


Figure 25: Operators network for $Q1$

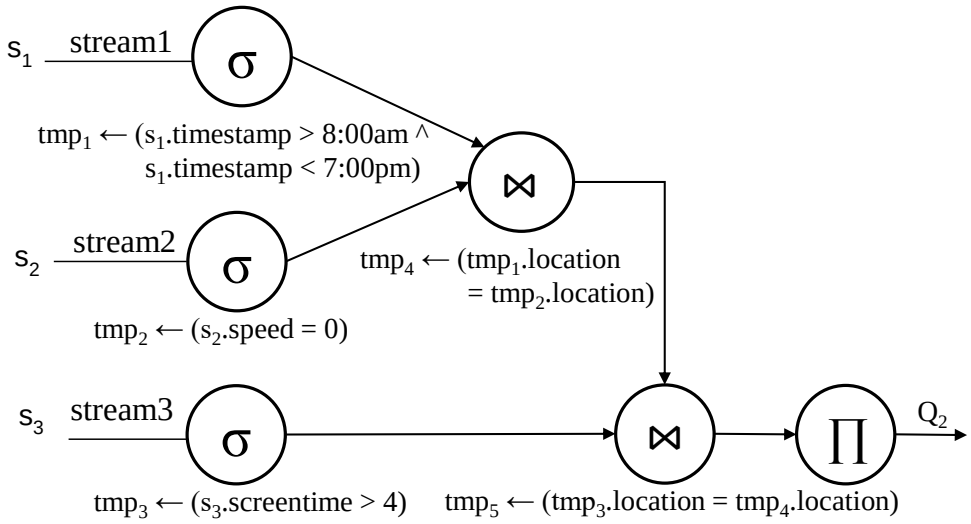


Figure 26: Operators network for $Q2$

t, which can be either time interval or tuple count. For example, in Q1 (Listing 5.2), the $\bowtie$ predicate that monitors the heart rate of users while driving between 6:00am and 7:00pm could be defined over a window of range 5 minutes and slide 1 minute: in the last 5 minutes every minute the stream id and heart rate of the user that matches the join condition will be reported.

The queries optimizer takes as input a set of queries $\mathcal{Q}$ and forms groups of queries that share the same sub-expressions. The optimizer then generates a shared-operator network,

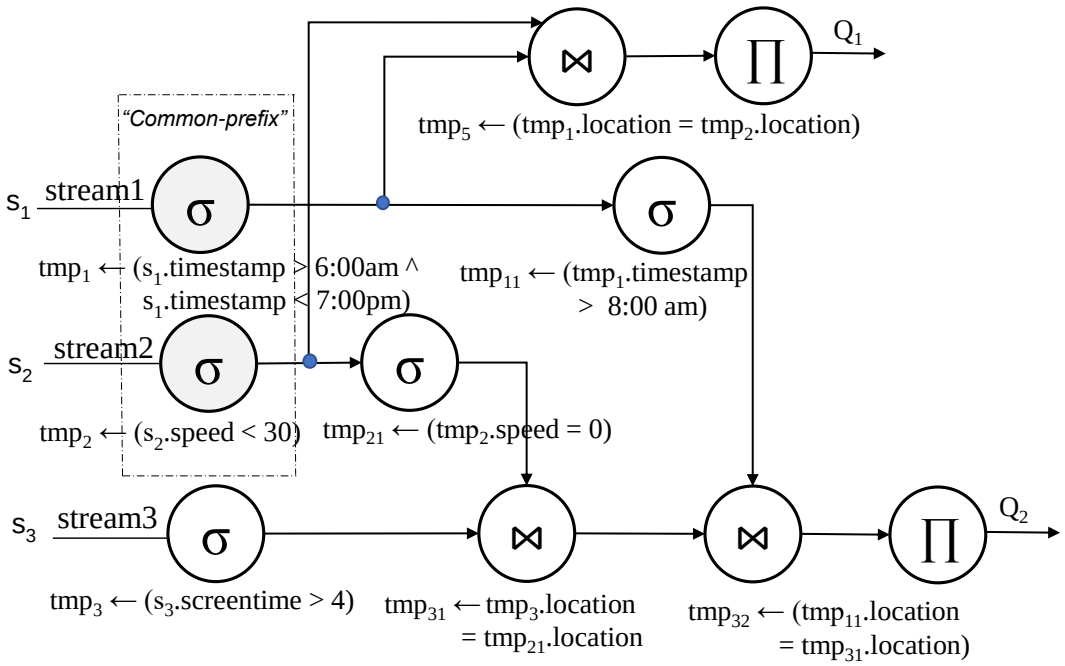


Figure 27: Shared-operators network for $Q1$ and $Q2$

arranged in a Directed Acyclic Graph (DAG) format, for each group of queries.

Definition 12. *An interleaved execution plan for a group of queries is a DAG network* $\mathcal{N} = (V, E, L)$. *V, E, and L being the set of vertices, edges and set of labels, respectively, and are defined as follows:*

- *A vertex* v_i *is introduced for every operator* o_i *in query* q_i. *If the results of* o_i *are used by more than one operator belonging to different queries, then the vertex* v_i *will be annotated as "Common-prefix"*
- *If the results of* o_i *are used in* o_j, *an edge* $(v_i \rightarrow v_j)$ *is introduced*
- *The label* $L(v_i)$ *is the processing done by the corresponding operator* o_i *(i.e.,* $operator_{predicate}$*)*

Figures 25 and 26 show the individual queries' operator networks, represented as DAGs, for the execution plans of $Q1$ and $Q2$, respectively, without window specifications, for clarity since they are the same for all three data streams. The incoming data stream tuples flow in the graph of operators to produce the final query results.

In this particular example, both $Q1$ and $Q2$ operate on the same input data streams and show a great overlap in their stream filtering conditions (both `SELECT` timestamps within the same interval and have the same `JOIN` conditions on the input streams). Therefore, a shared-operator network for these two queries can be constructed as shown in Figure 27. To generate this interleaved query execution plan, the standard multiple-query processing algorithm proposed in [128] was used given that the joins are over the same windows, range and slide. In the figure, the highlighted vertices are the ones that are annotated as "common-prefix" operators since those are the shared operators between both queries Q_1 and Q_2 (i.e., the output of these operators feed into two different operators belonging to the two queries).

5.2.3 ACCESS CONTROL MODEL AND SECURITY PUNCTUATIONS

There are different ways by which access control policies can be enforced in DSMSs. The assumption made here is that the DSMSs can enforce *query-based* access restrictions. Query-based access control policies can be specified by both the data stream providers and the DSMSs. For example, if a user does not renew his/her subscription to access certain query results in a DSMS, then this user would be temporarily denied access to any of his registered queries—by denying access to their corresponding data streams—until the subscription is renewed. Similarly, stream data providers can define policies to identify the different data consumers who are allowed to query those data streams (as shown in the example in Section 5.1).

Given the dynamic and heterogeneous nature of real-time data stream systems, DSMSs are likely to be using different access control policies and models. As such, the work presented here is very generic that it can accommodate a wide range of access control models (e.g., RBAC, ABAC, DAC) [58, 78, 125].

In general, let $\mathcal{P}$ denote the set of all authorization policies, each authorization policy $P : P \in \mathcal{P}$ enforced by a DSMS is defined as: $P \subseteq Q \times U$, where Q is the set of queries and U is the set of users or user roles/attributes. Let function m be a mapping such that $m : Q \rightarrow S$, that is, m identifies the set of data streams that are being accessed by a query.

To allow for query-based access restrictions to be enforced in the shared-operator net-

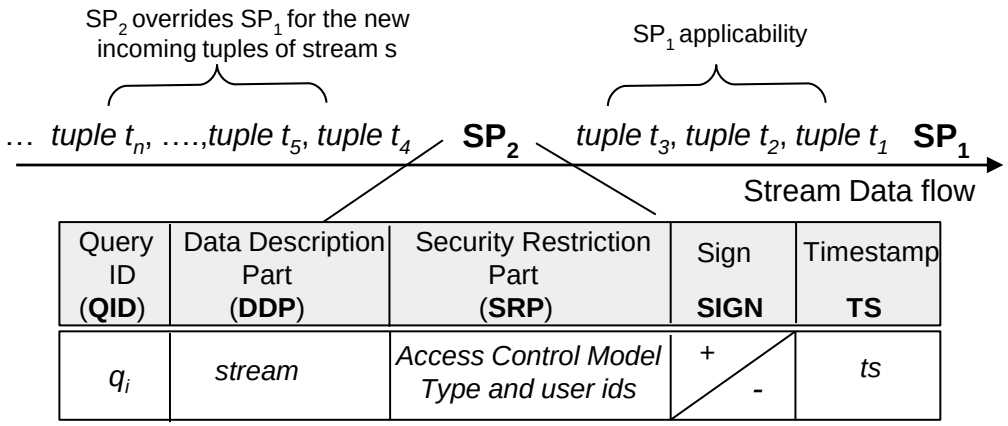


Figure 28: Security Punctuations injected in a data stream

works, an adaptation of *Security Punctuations* (SPs) [110, 109] is used. SPs are considered meta-data in the form of predicates that are injected into the data streams that describe the access control privileges of each query. SPs are embedded by the authorization servers and are used by the proposed algorithms to trigger the Privacy Switches to turn *on* or *off* in the shared-operator networks to enforce access control.

SPs are comprised of the following four fields (see Figure 28):

- *Query ID*: the identifier of the query that the SP is defining access for.
- *Data Description Part*: indicates the schema fields of a data stream tuple that are protected by the policy. This can be at the granularity of an entire stream, or specific tuples or attributes within the tuples. In the presented algorithm in Section 5.3, it is assumed that the access control applies to the entire data stream.
- *Security Restriction Part*: defines the access control model type and the data user(s) authorized by the policy.
- *Sign*: specifies if the authorization is positive (+) or negative (-).
- *Timestamp*: the time at which the SP is generated.

SPs are injected into the data streams in the order of their timestamps.

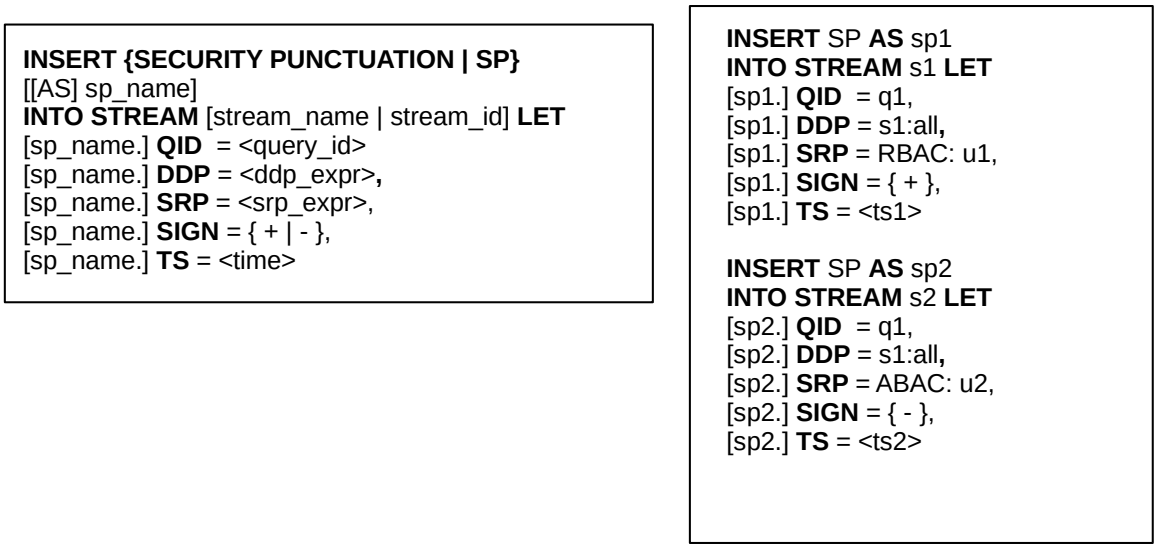


Figure 29: SP syntax (right) and sample SPs injected in two separate data streams (left)

For each authorization policy P, the authorization server will perform the following:

1. evaluate the proofs of authorization for each (q_i, u_i) pair in P
2. execute the mapping function m to identify the set of streams S that are involved in each query q_i in the case of query-based policies
3. construct the necessary SPs to identify the access privileges of each user or group of users and inject them in the corresponding streams

According to the outcome of the proofs of authorizations, the *Sign* field in the SP will be set to $Sign =$ '+', if a user may access the data stream tuples for a given query at any time $ts_{access} \geq ts$ or $Sign =$ '-' if a user is denied access to the data stream at any time $ts_{access} \geq ts$.

The authorization server is in charge of keeping track of the possible changes in the system. Possible changes include: new incoming users registering new queries in the system, changes in the access control policies, the changes in the users' credentials (e.g., revoke or expire). In case any of these changes take place, the authorization server re-writes the SPs corresponding to those changes to override them with a new "Sign" value and re-injects the

updated versions of the modified SPs into their corresponding data streams. Figure 28 shows one example of an input data stream with two different SPs injected in the stream as time evolves and new tuples arrive.

Figure 29 shows an extension to the CQL syntax [17] to support the specifications of SPs in data streams. The figure illustrates the syntax of an SP as well as an example of two SPs injected into two data streams. Each SP indicates the access privileges of a separate user over the tuples of the input data stream.

5.3 PRIVACY-AWARE SHARED-OPERATOR NETWORKS

This section describes in details the proposed approach to achieve a cost-effective way of handling access control in shared-operator networks. As mentioned earlier in Section 5.1, the naïve approaches of applying pre- or post-filtering may considerably limit the shared-operator network performance. The alternative approaches of constantly changing the interleaved shared-operator networks to isolate the queries that are no longer accessible by any of the users, or even maintaining different copies of shared-operator networks are both considered very costly approaches of handling access control.

To overcome these limitations, the proposed solution involves embedding a new set of operators called *Privacy Switches*(PrSs) in the networks. At a high level, the main idea is to strategically place these switches in the network to shut off branches of operators in the case of total access loss to certain query outputs or by filtering out the query results in the case of partial access loss to queries. By doing so, the shared-operator networks can execute disruptively and efficiently.

5.3.1 PRIVACY SWITCHES

Privacy Switches (PrSs) are the novel set of operators that will be integrated within the shared-operator networks. These switches allow shared-operator networks to execute the queries impeccably while performing access control-based filtering.

For this purpose, three different types of PrSs are introduced:

- **initial-switches**: placed at some of the input streams to each operator network and perform the traditional pre-filtering operations.
- **in-network switches**: embedded within the operator network and are capable of temporarily shutting off certain branches in the network to save the unnecessary execution of some operators in certain access permission cases.
- **terminal-switches**: placed at the query outputs and they act as multiplexers that can branch out the query output to multiple users.

All three types of PrSs operate just like any other conditional query operator. They can be viewed as, being similar to, `SELECT` or `PROJECT` operators' that filters data input streams based on the security predicates as determined by the SPs embedded in the streams. The location of the PrSs in the shared-operator networks is determined by a placement algorithm discussed in Subsection 5.3.2.

To better understand how the PrSs operate, assume there is a shared-operator network that interleaves the execution of multiple queries and each query could possibly be shared by multiple users in the DSMS. The three different types of the PrSs cover the following access scenarios:

- *Case I: partial loss of access*: In this case, only a subset of users lose access to the data streams processed by one or more of the interleaved queries in the shared-operators network. In this case, terminal switches will be in charge of granting access of the query results to the subset of users who did not lose their access privileges.
- *Case II: total loss of access*: In this case, all users lose access to one or more of the interleaved queries in the shared-operators network. Accordingly, instead of operating terminal switches and having possibly unnecessary operators executing, both initial and in-network switches will shut off the isolated branches of operators in the network that are not shared by any other queries. In this case, a considerable amount of unnecessary work will be saved and performance gains will be achieved.

Initial and in-network PrSs are characterized by the following attributes:

$$PrS =< Type, QueryID, AccessCounter >$$

The $Type$ attribute defines whether this is an initial or in-network PrS, $QueryID$ is the query that the PrS is governing access to, and $AccessCounter$ is a counter that changes during the execution of the PrS to identify the number of users who have access to that particular query. The counter will have a value greater than zero in the case of partial loss of access, and will be set to zero in the case of total loss.

Terminal switches are characterized by:

$$PrS =< Type, QueryID, U >$$

where U is the set of users that have access to the query $QueryID$ governed by this terminal switch.

Algorithm 5, shows the pseudocode for executing the different types of PrSs. The input to each PrS is a data stream with embedded SPs embedded. The PrSs execute this algorithm only when new SPs arrive in the data streams. The algorithm shows how the PrSs will allow or prevent the data streams from flowing through the network based on the output of the authorization predicates indicated by the *Sign* value in each SP.

Both initial and in-network PrSs operate in a similar manner. Each of these PrSs increment their *AccessCounter* whenever an SP with a matching QueryID and a '+' sign is encountered in the input stream and decremented each time a matching SP with a '-' sign shows up. In the case of partial access loss, the *AccessCounter* will be greater than zero (i.e., there is at least one SP in a stream that grants access to any user). In this case, the PrS will be switched *on* and the data stream tuples will flow normally through the network (lines 14 and 15). Note that the assumption made here is that the authorization server will re-inject SPs for the same user or set of users only if there are changes in the access control permissions for those users.

In the case of total loss of access, the *AccessCounter* will decrement down to zero (i.e., the last user who had access to the query lost that access). Accordingly, the PrS will be switched *off* and the flow of the tuples will halt temporarily (lines 16 and 17) until new SPs show up with positive access signs.

Algorithm 5: PrS execution algorithm

```
   input: Stream
 1 set PrS.AccessCounter = 0;
 2 new SPs_batch arrives;
 3 foreach SP ∈ SPs_batch do
 4 |  if SP.TS < ts_access then
 5 |  |  discard SP;
 6 |  else
 7 |  |  if PrS.Type = "in-network" OR "initial" then
 8 |  |  |  if SP.Sign = '+' AND PrS.QueryID = SP.QID then
 9 |  |  |  |  increment PrS.AccessCounter;
10 |  |  |  end
11 |  |  |  if SP.Sign = '-' AND PrS.QueryID = SP.QID AND
   |  |  |   PrS.AccessCounter != 0 then
12 |  |  |  |  decrement PrS.AccessCounter;
13 |  |  |  end
14 |  |  |  if PrS.AccessCounter >0 then
15 |  |  |  |  send stream to PrS output;
16 |  |  |  else
17 |  |  |  |  discard stream;
18 |  |  |  end
19 |  |  else
20 |  |  |  if PrS.Type = "terminal" then
21 |  |  |  |  if SP.Sign = '+' AND PrS.QueryID = SP.QID then
22 |  |  |  |  |  add SP.SRP.u_id to PrS.U;
23 |  |  |  |  |  discard SP;
24 |  |  |  |  |  send stream to output of SP.SRP.u_id;
25 |  |  |  |  else
26 |  |  |  |  |  if SP.Sign = '-' AND PrS.QueryID = SP.QID then
27 |  |  |  |  |  |  remove SP.SRP.u_id from PrS.U;
28 |  |  |  |  |  |  discard stream;
29 |  |  |  |  |  end
30 |  |  |  |  end
31 |  |  |  end
32 |  |  end
33 |  end
34 end
```

Terminal switches operate slightly different. They multiplex the final query output tuples to the users that have positive access as defined by their SPs (lines 20 - 24). Note that for privacy preservation, the default setup of terminal switches is deny the output to all users and only grant access when explicitly granted by an SP (i.e., they start by an empty set of

users U). Similarly, initial and in-network switches have their *AccessCounter* initialized to zeros, which by default will deny any access to the query outputs.

The operation of the PrSs is very similar to the well known notion of counting semaphores that are typically used by many systems to coordinate access to different resources. By looking at the status and counter of each PrS, the DSMSs can collect statistics about how many users in the system are currently allowed access to a certain query output. The PrS present an effective and simple solution for enforcing the access control in shared-operator networks. They allow the on-the-fly adjustment of the network status as changes in access control take place without the need to re-direct the input streams to different networks or the need to restructure the operator network. The solution not only preserves the performance benefits gained by the shared-operator networks but also ensures that during the execution of the queries, no data tuples will be dropped or discarded as the network adjusts itself to the access control changes, which in turn preserves the precision of the query output results.

5.3.2 PLACEMENT OF PRIVACY SWITCHES

This subsection will explain the placement choices of all three types of PrSs in shared-operator networks, especially of the in-network ones. Algorithm 6 shows the pseudo-code for the PrSs placement algorithm. This placement algorithm extends the query optimizer, i.e., after a query optimizer constructs a shared-operator network strategically embedding the switches in the network.

The input to the algorithm is a shared-operator network pre-computed by the queries optimizer in the DSMSs. The placement of terminal switches is straightforward, at the end of each query output a terminal-switch will be inserted (line 2). In-network switches placement requires some analysis of the network graph. The main idea is to find the operators that are shared by multiple queries. Those shared-operators annotated are the ones annotated as "Common-Prefixes". The in-network switches will be placed along the outgoing edges of the last set of operators that belong to the common-prefixes (line 15). Finally, the initial-switches are placed at the input streams (lines 7 and 8) to apply pre-filtering of the input streams in the case of total access loss.

Algorithm 6: PrSs placement algorithm

input: A shared-operator network $\mathcal{N} = (V, E, L)$
Data: s defines a stack

```
1  foreach query q_i output traverse N backwards (DFS-search) do
2      insert PrS = <Type = "terminal", QueryID = q_i, U = null > at the
        output of q_i;
3      s.push(v);
4      while s is not empty do
5          v = s.pop();
6          if v is NOT annotated "Common-prefix" then
7              if N.adjacentEdges(v)==φ then
8                  insert PrS =
                    <Type = "inital", QueryID = q_i, AccessCounter = 0 > at the
                    input stream;
9              else
10                 foreach edge from v to w ∈ N.adjacentEdges(v) do
11                     s.push(w);
12                 end
13             end
14         else
15             insert PrS =
                <Type = "in − network", QueryID = q_i, AccessCounter = 0 > at
                outgoing edge from v;
16         end
17     end
18 end
```

5.3.3 EXAMPLE EXECUTION OF PRIVACY SWITCHES

In this subsection, the motivating example from Section 5.1 will be revisited to ilustrate the behavior of the shared-operator networks with the PrSs in place. Figure 30 shows the same shared-operator network with the PrSs embedded in the network according to the proposed placement algorithm.

Assume that the shared-operator network from Figure 27 is being executed by multiple users who initially have access granted to all three data streams, hence all users can view the output of both queries. From Figure 30, the dotted box highlights the "Common-prefix" zone of the operators shared by both executing queries. According to the switches placement Algorithm 6, the in-network privacy switches are placed right after those operators. Also

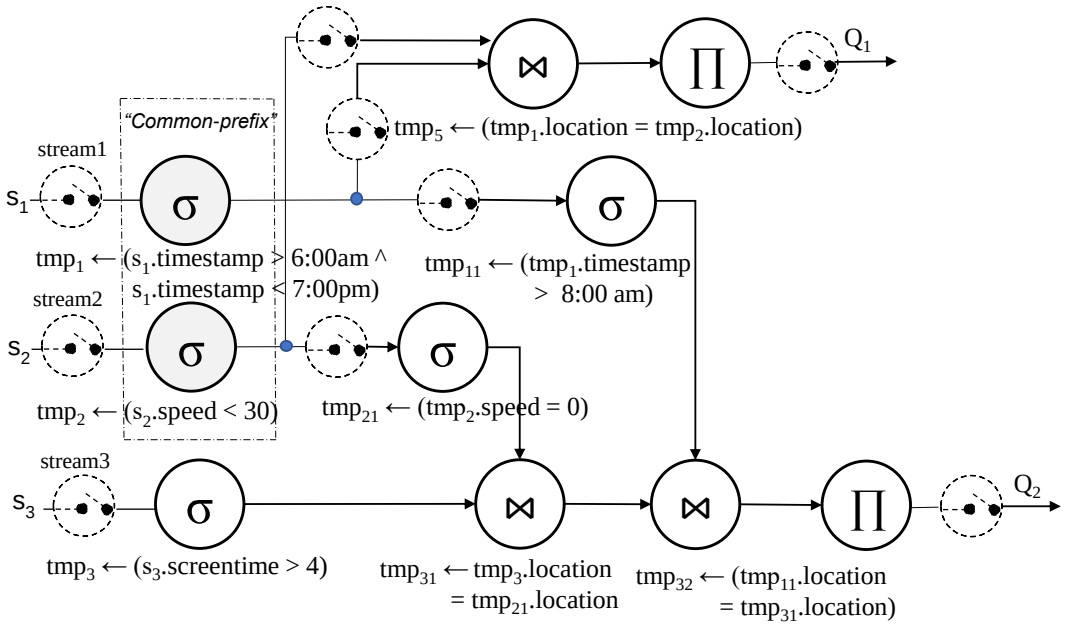


Figure 30: Example of a privacy-aware shared-operators network

at the output of both queries, terminal switches are placed, and at the front of the streams initial PrSs are placed.

In this particular example, if all users accessing $Q2$ lose access to the input streams feeding into this query, the PrSs will shut off six out of the ten operators in the network saving unnecessary processing and bandwidth. When any of these users gain their access back, the PrSs will resume operating all the nodes back. This shows that all three types of switches orchestrated together are capable of enforcing the access control policies in shared-operator networks in a cost-effective way.

5.3.4 CORRECTNESS OF PRIVACY SWITCHES

This section will present proof of correctness of the PrSs operation to ensure that the data streams privacy are enforced at all times. The proof will show that the PrSs will only allow tuples of a particular stream to be accessed by the users that are specified in the SPs, otherwise they will enforce denial-by-default.

Theorem 3 (PrSs Correctness)**.** *During the execution of the PrSs as described by Algorithm 5, for any tuple t, PrS will allow the flow of t only if its preceding SP has a '+' access sign, otherwise PrS will block the flow of t.*

Proof. To prove this claim, the following must be asserted:

(i) terminal PrSs will only allow users specified by the SPs to access the tuples.

(ii) in-network and initial PrSs only allow tuples to flow through the network if there is at least one user that has access granted.

Note that the terminal PrSs are the switches that have the ultimate control of which users can access the tuples of a particular data stream, even if the initial and in-network PrSs allow all the tuples to flow through the network. As such, proving the privacy of a shared-operators network is only dependent on proving that assertion (i) is true. Yet, to prove that a shared-operators network not only denies access to the data streams to those users who are not allowed access, but also ensures that access is granted to those users who are allowed access, then both assertions (i) and (ii) need to be true.

For the base case, assume that $\exists$ authorization policy P that specifies that u_1 can access q_1 and SP is used to encode this policy and is injected in all data streams used in processing q_1. Let $t \in T$, where T is a set of tuples, be a tuple that belongs to the data stream used in processing q_1. There are two cases to be considered to assert both (i) and (ii):

1. SP arrives prior to tuple t,
2. tuple t arrives prior to SP,

Case 1: if $SP.q_1$ is '+', then all initial and in-network PrSs processing this SP will increment their $PrS.AccessCounter$ and allow the following data tuple t and $SP.q_1$ to reach to the terminal switches. Terminal switches will in turn add $SP.SRP.u_{id}$ to the list $PrS.U$ and open up the access channel for this user to allow tuple t to flow into query output.

If $SP.q_1$ is '-', then all initial and in-network PrSs will decrement their counters. If the counter value goes down to zero, this means this user was the last user to have access and the PrSs will not allow the tuples to flow to the output (total loss case). If the counter is greater than zero, then the tuples will flow to the terminal switches. For each terminal

switch, if $SP.SRP.u_{id} \in PrS.U$, then this user will be removed from the list, and the access channel to this user will be blocked.

Case 2: tuple t will only flow to the channel assigned to user u_1 iff $u_1 \in PrS.U$. Since users are only added to the PrS users list and given access if an explicit SP with a '+' sign is encountered at time $SP.TS < ts_{access}$, then if t arrives prior to the SP, the denial-by-default will be enforced.

The above cases account for the possible scenarios of a set of tuples T and their equivalent SPs showing up in the data streams. By proving that both assertions (i) and (ii) are true, it is shown that Theorem 3 holds in the base case. □

Observe that an argument similar to that used in the base case shows that the same behavior of PrSs would apply to all policies. Furthermore, the inductive hypothesis can be used to prove that the value of the $SP.Sign$ is in charge of activating or deactivating the users channels of the terminal PrSs. As such, only those tuples that are preceded with a positive SP sign will flow to the users specified by the SPs, and Theorem 3 holds for all policies.

5.4 EVALUATION

The following sections will present the details of the random shared-operator networks generator used in the performance evaluations of the proposed privacy-aware shared-operator networks as well as the experimental results.

5.4.1 CONFIGURATIONS AND EXPERIMENTAL SETUP

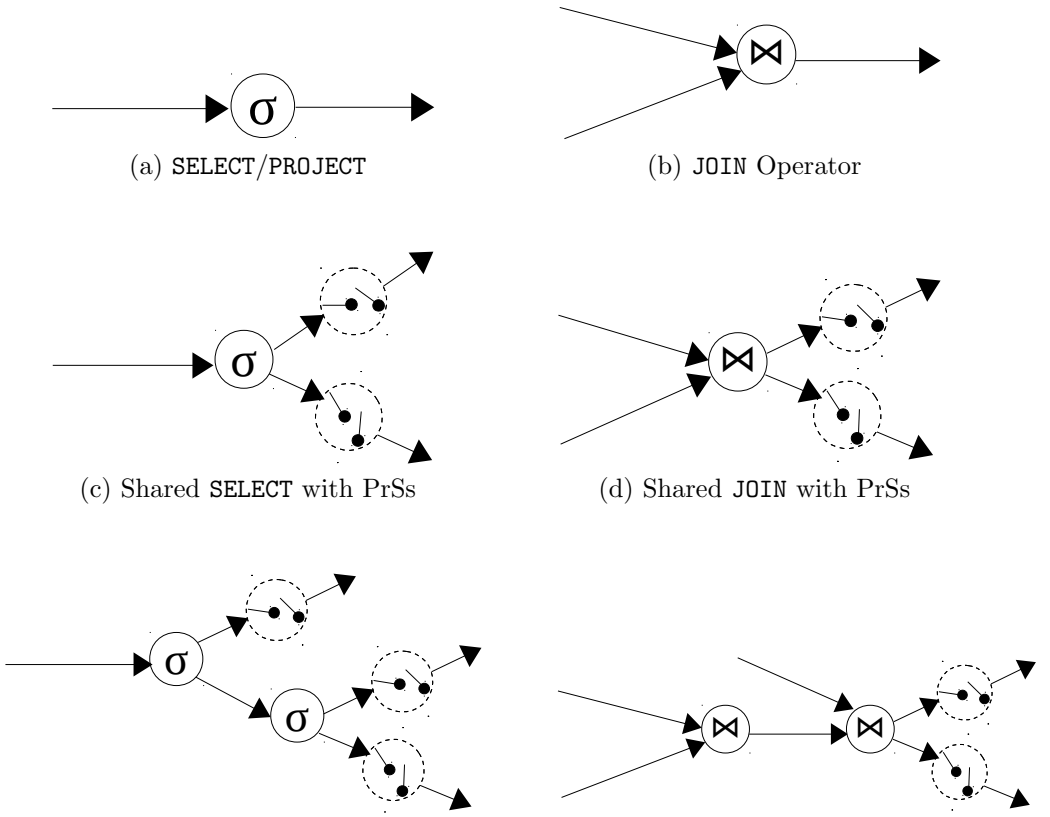
Generating synthetic shared-operator networks gives control over the input parameters and the different scenarios of access control. To implement the simulator, the jGraphT Java library was used to generate DAGs that mimic networks of interleaved shared-operators. The simulator takes as input the following parameters: *number of input streams*, *number of interleaved queries*, *number of users executing those queries*, *number of query operators*, and *degree of sharing*. The simulator given these parameters generates DAGs of nodes of the following types:

- *`SELECT` nodes*: One input and one output
- *`JOIN` nodes*: Two inputs and one output
- *Shared `SELECT` nodes*: One input and two outputs
- *Shared `JOIN` nodes*: Two inputs and two outputs

The degree of sharing input parameter indicates the percentage of the query operators that will be included in the "common-prefix" of the network (i.e., how many query operators will be shared across the executing queries). An assumption is made that all shared operators are defined over the same window specifications which are omitted for simplicity of the analysis. Based on the total number of query operators and the degree of sharing, the simulator randomly generates the query operators. The interconnections between the query operator nodes is randomly chosen by the simulator as the DAG is being built. Some rules during the generation process were enforced to assure the correctness and validity of the generated networks. For example, the "join-push-down" approach of operators was enforced (i.e., `SELECT` and shared `SELECT` nodes appear before `JOIN` and shared `JOIN` nodes). This is a common practice for queries optimizer to execute the `SELECT` operators first to filter out the tuples as early as possible and improve queries processing times.

After the random shared-operator networks are generated, the placement Algorithm 6 executes to identify the locations of all PrSs. The number of PrSs is dependent on the topology of the generated network. Finally, the users are randomly assigned to the query outputs of each generated graph.

Figure 31 shows a sample of the different possible configurations of query operators as



(a) SELECT/PROJECT

(b) JOIN Operator

(c) Shared SELECT with PrSs

(d) Shared JOIN with PrSs

(e) Consecutive shared SELECT operators with PrSs

(f) Consecutive shared JOIN operators with PrSs

Figure 31: Different variants of randomly generated operators

generated by the simulator. Figures 32 and 33 illustrate two randomly generated shared-operator networks by the simulator. Both random networks were generated with the following parameters: input streams = 3, queries = 3 and total operators = 12. The degree of sharing for the first network was set to 20%, while for the second network was set to 80%. In the figures, the highlighted gray operators are shared and belong to the "common prefix".

Note that the final total number of query operators generated could be slightly higher than the initial input parameter. The reason for this is, that in some cases, and due to the randomness of the interconnections between the nodes, the generated topology might have fewer or larger number of query outputs than the desirable input parameter. In these cases, the generator introduces some extra nodes to the DAG. For example, in the case of

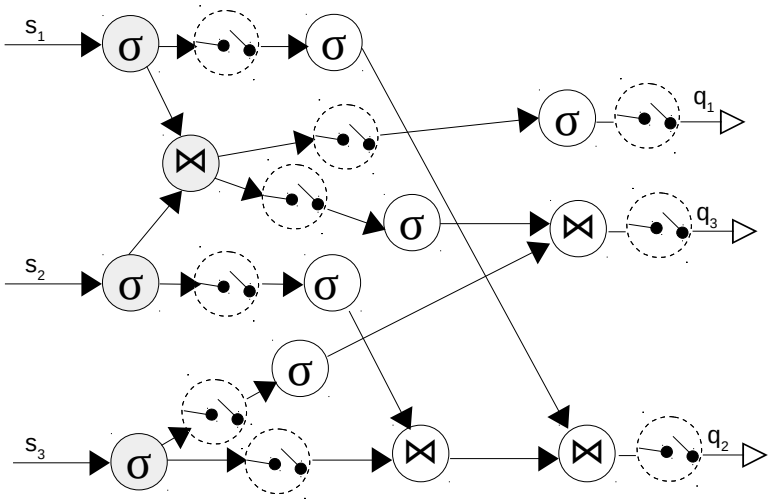


Figure 32: Sample 1: randomly generated operators network

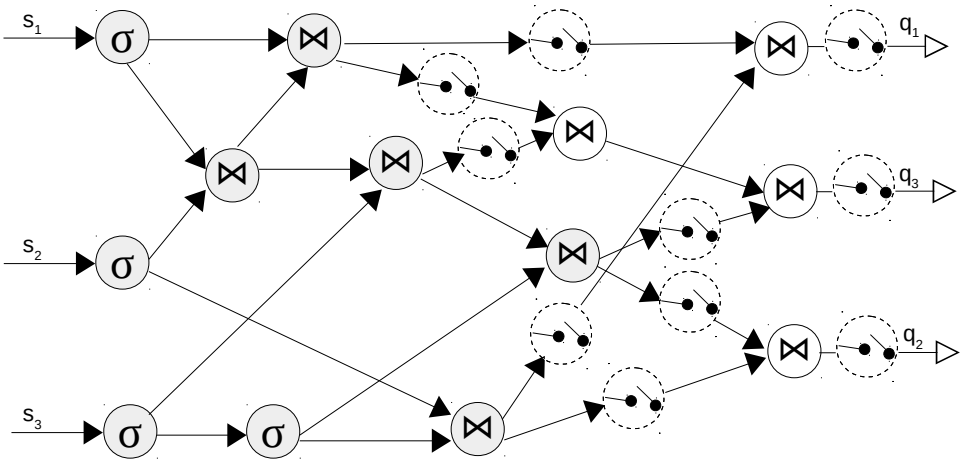


Figure 33: Sample 2: randomly generated operators network

fewer query outputs, the generator inserts a new branch of nodes that starts with a shared `SELECT` operator and ends with a new query output. In the case of larger query outputs, the generator combines the query outputs using a `JOIN` node.

In the experiments, the average processing time per tuple for each query operator in the network was set to 0.1ms, and the average processing time per SP for each PrS to 0.1ms

as well. Since these simulations are intended to compare and contrast the performance of different operator networks, these times were chosen as rough estimates and defined as constants throughout the execution of the networks. In reality, the processing times of operators would be different from one another.

The upper bound on the processing time of each network to process 1000 input tuples was computed by adding up the number of executing nodes in each network and multiplying that by the processing times for each tuple. Note that the reported times are upper bound since not every single operator in the network will process all 1000 tuples, as the input tuples get filtered by the `SELECT` operators, fewer tuples will be processed in the network. The arrival rate of SPs was set to be 1000 tuples (i.e., for every 1000 tuples, new SPs will show up in the input streams for each user in the system). The PrSs will process all SPs but will only take action for those SPs that match with the $SP.QID = PrS.QueryID$.

To better understand the security enforcement overheads in different shared-operator networks, the experiments were performed using varying parameters and conditions of the networks setup. In the experiments, the upper bound of the computed execution times were compared for the following three cases:

- *no-sharing* – each user is executing a separate operator network for each submitted query (base case),
- *shared without PrSs* – shared-operator network without the security enforcement PrSs (only post-filtering applied),
- *shared with PrSs* – shared-operator network with security enforcement using PrSs.

5.4.2 EXPERIMENT1: VARYING DEGREE OF SHARING

These set of experiments attempt to answer the following question:

Q1: what are the security enforcement overheads in the shared-operator networks induced by the PrSs?

To answer this question, the experiments computed the average processing times of 1000 input tuples as the degree of sharing between queries changes. The experiments were executed twice for different input parameters, once for 3 interleaved queries for a total of

15 operators and another for 5 interleaved queries for a total of 25 operators. The degree of sharing was varied, and for each degree the network execution times were averaged for 10,000 randomly generated operator networks.

Figures 34 and 35 show the reported execution times (in log_2 ms) of the networks comparing shared-operator networks with and without PrSs to the non-shared networks. The figures show that the shared-operator network outperforms the non-shared networks. The average execution time savings between the non-shared and shared networks was approximately 95%. This validates the benefits of executing shared-operator networks in DSMSs. The experiments show that the PrSs add negligible overheads to the network performance. Shared-operator networks with PrSs showed an average of 0.5% increase in the execution time than shared-operator networks without PrSs. This behavior is justified by the fact that PrSs execute infrequently (only when they encounter a new SPs in the streams that match the query id). Note that in reality, the execution of PrS would be similar in cost to `SELECT` operators, since they are only checking and updating their *AccessCounter* values based on the *Sign* values of SPs.

The experiments also show that the number of switches inserted in the networks by the placement algorithm were insensitive to the degree of sharing, similarly the execution times of the networks were also insensitive to the degrees of sharing unlike the non-shared versions of the network. These experiments showed that PrSs do not induce any considerable overheads to the execution times of the networks in the case of full access grants to all users in the system.

5.4.3 EXPERIMENT2: VARYING ACCESS CONTROL

These experiments were designed to answer the following two questions:

Q2) *How much cost savings in the networks can be achieved as users start losing access to queries*? and

Q3) *How much does the overlap between queries affect the execution times as users start losing access to queries*?

To answer both questions, the experiments measured the system performance (in terms

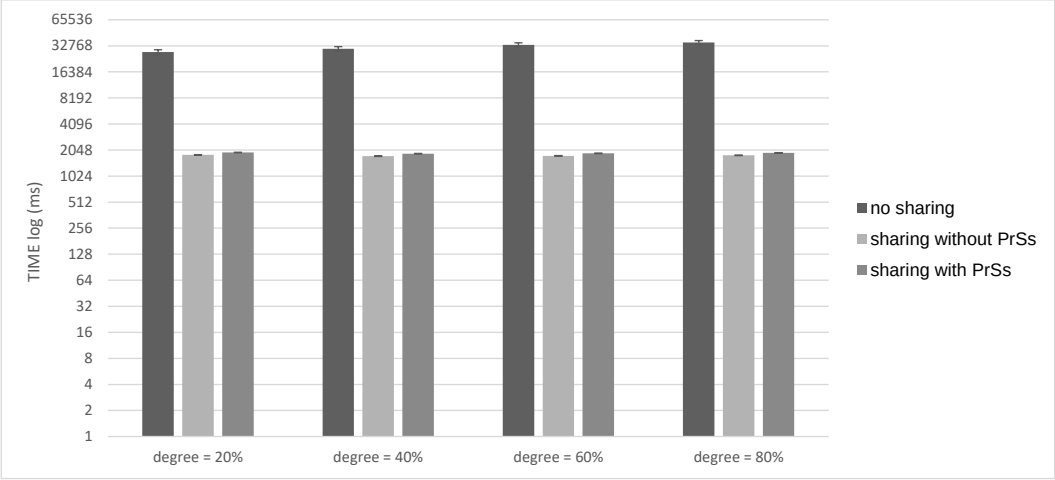


Figure 34: Network execution time for a shared-operator network size of 15 operators

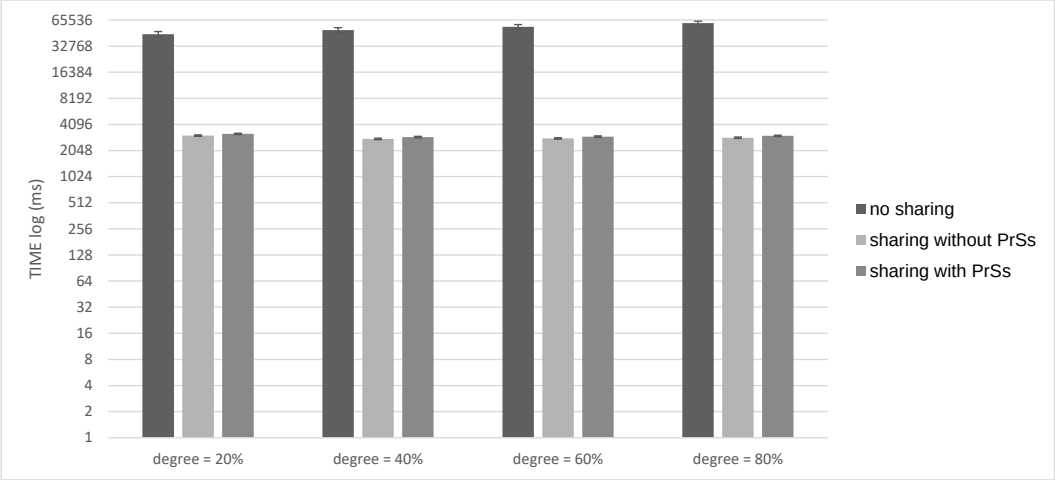


Figure 35: Network execution time for a shared-operator network size of 25 operators

of execution times) as users start losing access to query outputs. To cover all different cases, all different possible combinations of access loss were examined. The degrees of sharing in the network were varied to see the effect the degree of sharing among queries has on the total performance. For each degree of sharing examined, the network execution times were averaged for 10,000 randomly generated shared-operator networks. For each generated

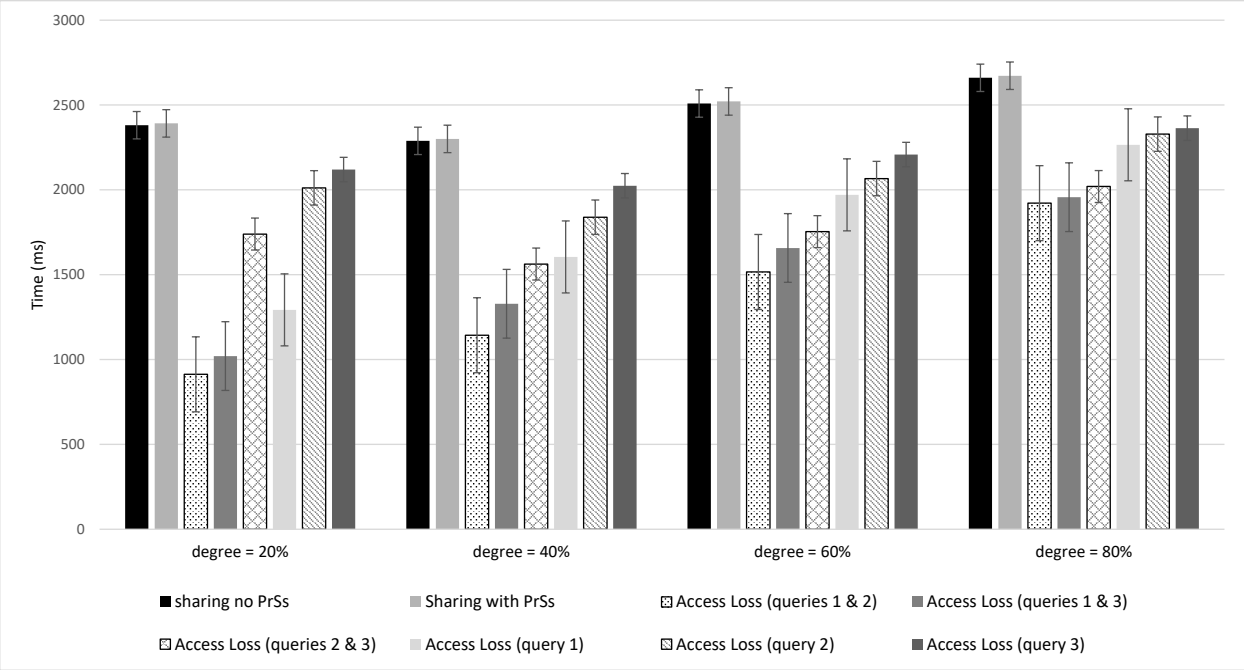


Figure 36: Network execution time for varying degrees of sharing and access loss patterns

network, the execution times were measured in the following scenarios: shared network without PrSs and only post-filtering, shared network with PrSs and full access to all queries, shared network with PrSs and all possible combinations of queries' losses. Figure 36 shows the results for networks generated with 3 input streams, and 3 interleaved queries with an average total of 20 query operators. On average 11 PrSs were inserted in the generated graphs.

From Figure 36, and consistent with the previous experiments, the difference in execution times between shared networks with and without PrSs was minimal. As the queries start losing access, the savings in the execution times were noticeable when compared to the cases of shared with only post-filtering of query outputs and shared with PrSs and full access. The average execution time savings between the shared with PrSs and shared without PrSs was approximately 38% in the case of two out of three queries lose access, and 19% in the case of one query access loss.

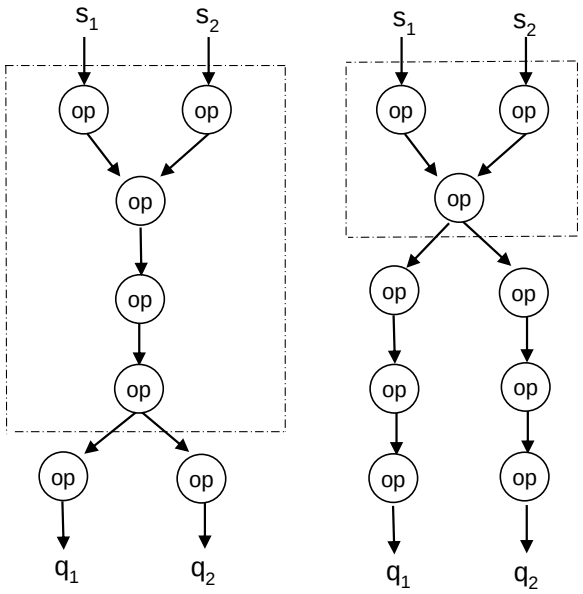


Figure 37: Examples of two different degrees of sharing in shared-operator networks

Another observation is that PrSs improve the network performance when the queries have lower degrees of sharing (i.e., less common subexpressions). The reason is, with fewer shared-operators, the in-network switches can shut off more isolated (unshared) query operators in the case of total users loss of some query outputs. To better illustrate this observation, consider Figure 37 that represents two shared-operator networks. The network to the left has a higher degree of sharing than the one to the right (almost 70% sharing vs. 30% sharing). The operators inside the dotted box represent the "common-prefix" shared operators. In the case of total access loss to either q_1 or q_2, the left network will execute all the shared operators at all times and only shut off the ones that are non-shared, which is in this case only one operator. On the other hand side, the network with less sharing (to the right), will shut off more operators in the case of total loss.

This behavior of the privacy-aware shared operator networks show that even with low degrees of sharing, the performance benefits would be even bigger when intermittent total access control loss is encountered.

5.5 SUMMARY

In this chapter, a novel technique for enforcing access control over shared-operator networks in DSMSs was introduced. The technique presented allows DSMSs to interleave the execution of multiple overlapping queries that are shared by multiple users in the system while applying access control restrictions on a per user or group of users basis without inducing any modifications to the shared-operator networks. The solution integrates new operators, Privacy Switches (PrS), into already existing shared-operator networks. The PrSs are capable of seamlessly configuring the network to allow or deny access to certain query results complying with the access control policies defined in the system.

The desirable balance between all 3Ps requirements of distributed systems were evident in the proposed solution. The solution presented ensures that access control policies will be enforced at all times even when frequent changes to the access control policies are encountered. Data precision is also guaranteed by the proposed solution. Since, the PrSs allow the shared operator network to execute uninterruptedly while handling changes in the access control, this ensures that no datastream tuples will be lost or dropped and hence query results are never affected. Finally, from the experimental evaluations, it was shown that the proposed technique induces minimal overheads on the shared-operator networks while achieving great gains in the network performance in the cases of intermittent access loss to some streams and queries. The technique proved to perform better in the case of total access loss with queries that have less common subexpressions, which shows greater benefits of the privacy-aware shared-operator networks even in lower degrees of sharing.

While the cost metric used in the evaluations was focused mainly on the execution times of the networks in different scenarios, the performance gains extend to more than that. Having a dynamically configurable shared-operator network saves not only time, but also bandwidth consumption, and more importantly all the consequential monetary costs associated with configuring and executing shared networks operating in cloud environments.

6.0 RELATED WORK AND ALTERNATIVE APPROACHES

Existing work in literature has proposed solutions to a subset of the problems addressed in this dissertation. This chapter will highlight the bodies of work most directly related to the three classes of representative distributed systems studied in this dissertation. Section 6.1 introduces the previous research in wireless sensor networks focusing on three main aspects of designing wireless networks, namely: data aggregation, privacy and fault-tolerance. Section 6.2 discusses the related research work pertaining to cloud solutions focusing on the properties that were examined in this dissertation, namely: consistency, privacy and distributed authorizations and distributed transactions. Finally, Section 6.3 discusses the related work to the third representative system, Data Stream Management Systems (DSMSs) focusing on the following properties: query optimizations and access control to stream data.

The solutions proposed for the distributed systems discussed in this dissertation are a result of a number of design choices that attempts to balance all 3Ps. Section 6.4 summarizes how this dissertation addressed a new unexplored space in each of the three representative systems.

6.1 WIRELESS SENSOR NETWORKS

There has been extensive work on data aggregation schemes including (e.g., [56, 135, 101]). These schemes assume both a reliable and a secure network. However, in the real world, sensor nodes are usually deployed in hostile environments where communication links can be an easy target for adversarial eavesdropping. The reliability assumption is also a strong one, as sensor networks are subject to frequent link failures that might cause the loss of a whole

sub-tree aggregate value. Subsections 6.1.1 and 6.1.2 will present the previous works that addressed either privacy-preserving in-network aggregation or fault-tolerant communications in wireless sensor networks.

6.1.1 PRIVACY PRESERVING SYSTEMS

A simple approach to partially preserving the confidentiality of aggregated data in WSNs is using hop-by-hop encryption [38, 152]. The problem with all hop-by-hop encryption schemes is that the confidentiality of the data is lost at any compromised node since the data is decrypted before the aggregation. Besides the confidentiality violation at nodes, the number of necessary shared secret keys becomes a function of the network density where all neighboring sensor nodes must share secret keys.

A higher level of confidentiality is achieved through complete end-to-end encryption of the sensor readings and their aggregate values. The use of homomorphic cryptosystems such as RSA [121], ElGamal, Elliptic Curve [106], or Paillier [113], would allow for end-to-end encryption with in-network aggregation. Unfortunately most such algorithms require extensive computations and very long keys that do not suit the computational and power constraints of WSNs. To enable additive aggregations of sensor values, the protocol presented in this dissertation (Chapter 3) makes use of the additively homomorphic symmetric-key stream cipher proposed by Castelluccia et al. in [33], which is a simple and efficient cipher system and well-suited for use in WSNs.

Existing confidentiality-preserving aggregation protocols (e.g., [19, 116]) share one common assumption: no packets are lost during the aggregation process,which is not a valid assumption in many WSNs. In some proposed schemes such as [74], additive aggregation is supported using a secret splitting technique for the sensor readings. In such scheme, a single packet loss would cause the whole aggregation process to fail and the inability of the sink node to recover a precise aggregation result. The authors in [19] proposed an end-to-end encryption for sensor network traffic, but their scheme is not resilient against link failures. Their proposed scheme utilizes encrypted default values to compensate for link failures, which does not solve the problem of delivering the correct values when link failures happen.

6.1.2 FAULT TOLERANCE MODELS

In typical in-network data aggregation systems, a query disseminates from a sink node to all other sensor nodes. As the query propagates in the network, a spanning tree is constructed. The spanning tree is rooted at the sink node, and each intermediate sensor node receives values from its children, aggregates them with its own value, and forwards the result to its parent. One major drawback of spanning trees is that they are not robust against link failures: a single link failure causes the value of the sub-tree connected by this link to be lost. Link failures that occur in the upper levels of a tree can thus cause massive losses of data. To address this problem, existing mechanisms (e.g., [26, 108]) make use of multipath in-network aggregation. This approach adds more robustness against link failures, but at the same time introduces challenges with correctly handling duplicate-sensitive aggregates such as SUM, AVG, and COUNT.

Different variations of so-called *sketches* provide approximate aggregation in WSNs for duplicate insensitive queries (e.g., [61] and are based on FM-sketches [59]). Unfortunately, these solutions are not applicable because aggregate computations are infeasible when using encrypted sketches.

In [64], RideSharing is introduced as a fault-tolerant scheme for duplicate sensitive aggregations in WSNs. RideSharing has been shown to outperform other fault tolerant schemes (e.g., Synopsis Diffusion [108]), consuming up to 50% less energy and bandwidth resources, while delivering more accurate aggregate results.

6.2 CLOUD DATABASE TRANSACTIONS

Many database solutions have been written for use within the cloud environment. For their cloud infrastructure, EC2 [1], Amazon uses their own database solution called Dynamo [51], which is built on top of their S3 [2] storage layer and is motivated by a desire to provide high availability among thousands of servers. Google built Bigtable, which is widely used for their own services such as Google Earth, Google Finance, and their web indexes [40]. Facebook

implemented Cassandra, which is now maintained by Apache, which implements a simple key-value store model [88]. Yahoo!'s PNUTS [47] is a massively parallel and geographically distributed database system that serves Yahoo!'s web applications.

The fact that these new database projects were implemented even though mature database solutions were already available suggests that the cloud environment requires a level of specialization not before seen. Therefore, there has been a plethora of research work that has been published generally on cloud computing and particularly on management cloud databases. The following sections will review some of the research work that is orthogonal to the work presented in this dissertation.

6.2.1 CONSISTENCY MODELS FOR THE CLOUD

Distributed database systems for cloud applications emphasize scalability, fault tolerance and availability, sometimes at the expense of consistency or ease of development. To achieve a balance between these requirements, a certain number of replicas of the data has to be maintained at different locations. When data is replicated over a wide area, consistency part of the common ACID (Availability, Consistency, Isolation, and Durability) properties is typically compromised to yield reasonable system availability.

Such a relaxed consistency model adds a new dimension to the complexity of the design of large scale applications and introduces a new set of consistency problems [144]. Luis Vaquero et al. [142] propose a tree-based consistency approach that maximizes the consistency and performance. The tree is formed between primary server to all replica servers in such a way that the maximum reliable path is ensured between them, which reduces the probability of transaction failure. In [71], the authors presented a model that allows queriers to express consistency and concurrency constraints on their queries that can be enforced by the DBMS at runtime. On the other hand, [85] introduces a dynamic consistency rationing mechanism which automatically adapts the level of consistency at run-time. All previous research work so far focuses on *data* consistency, while the work presented in this dissertation focuses on attaining both *data* and *access control policy* consistency [91, 89, 90] to deliver an interesting balance between the 3Ps.

6.2.2 RELIABLE OUTSOURCING

Security is considered one of the major obstacles to a wider adoption of cloud computing. Particular attention has been given to client security as it relates to the proper handling of outsourced data. For example, proofs of data possession have been proposed as a means for clients to ensure that service providers actually maintain copies of the data that they are contracted to host [21]. In other works, data replication has been combined with proofs of retrievability to provide users with integrity and consistency guarantees when using cloud storage [27, 134].

To protect user access patterns from a cloud data store, Oblivious RAM (or ORAM for short) [49, 99, 68], originally proposed by Goldreich and Ostrovsky [66] is used to access encrypted data residing on untrusted storage servers, while completely hiding the access patterns to storage. Particularly, the sequence of physical addresses accessed is independent of the actual data that the user is accessing. To achieve this, ORAM continuously re-encrypts and reshuffles data blocks on the storage server, to cryptographically conceal the logical access patterns while allowing users to issue encrypted reads, writes and inserts [148]. Further, [149] proposes a mechanism that enables untrusted service providers to support transaction serialization, backup, and recovery with full data confidentiality and correctness. This work is orthogonal to the problem addressed in this dissertation, namely solving consistency problems in policy-based database transactions that can lead to perilous privacy problems and its interplay with data precision and transactions performance (discussed in Chapter 4).

6.2.3 DISTRIBUTED TRANSACTIONS AND AUTHORIZATIONS

Executing transactions in the cloud is not a new revelation. It is assumed that strict transactions are still necessary for many applications in the cloud. The work presented in CloudTPS [147] provides full ACID properties with a scalable transaction manager designed for a NoSQL environment. To maintain the ACID properties even in the case of server failures, the data items and transaction states are replicated to multiple local transaction managers. Periodically consistent data snapshots are checkpoint to the cloud storage service.

Consistency correctness relies on the eventual consistency and high availability properties of cloud computing storage services. However, CloudTPS is primarily concerned with providing consistency and isolation upon data without regard to considerations of the issues that arise when authorization policies follow the same eventual consistency model.

There has also been recent work that focuses on providing some level of guarantee about the relationship between data and policies [150]. This work proactively ensures that data stored at a particular site conforms to the policy stored at that site. If the policy is updated, the server will scan the data items and throw out any that would be denied based on the revised policy. It is obvious that this will lead to an eventually consistent state where data and policy conform, but this work only concerns itself with local consistency of a single node, not with transactions that span multiple nodes.

The consistency of distributed proofs of authorization has previously been studied, though not in a dynamic cloud environment (e.g., [91]). This work highlights the inconsistency issues that can arise in the case where authorization policies are static, but the credentials used to satisfy these policies may be revoked or altered. The authors develop protocols that enable various consistency guarantees to be enforced during the proof construction process to minimize these types of security issues. These consistency guarantees are similar to the proposed notions of safe transactions (Section 4.2. However, the work presented in this dissertation also addresses the case in which policies—in addition to credentials—may be altered or modified during a transaction.

6.3 DATA STREAM MANAGEMENT SYSTEMS

The third representative system in this dissertation focuses on Data Stream Management Systems (DSMSs). From the data stream community, many data streams processing systems have been developed both academically (e.g., Aurora [13], Borealis [29], STREAM [16], TelegraphCQ [39]) and commercially (e.g., IBM Streams [7]). These systems differ in many ways than the traditional Database Management Systems (DBMs). The main differences rely on the fact that DSMSs are very dynamic and they deal with ubounded continuous data sets

and continuous queries. Therefore, DSMSs introduce a new set of challenges pertaining to the optimized executions of the continuous queries as well as to the privacy concerns of data sensitive stream applications. This section will highlight the related work that addressed these two sets of challenges.

6.3.1 OPTIMIZED QUERY PROCESSING

One of the major challenges that face processing queries for data streams is optimizing the processing and placement of multiple continuous queries which is imperative for DSMSs to reach their full potential.

Early work on multi-query optimization includes work by Sellis [128], Park and Segev [114], and Rosenthal and Chakravarthy [123]. The work of Shim et al. [132] considers heuristics to reduce the cost of multi-query optimization. However, even with heuristics, these approaches are extremely expensive for situations where each query may have a large number of alternative evaluation plans. Extensive research in the area yielded more cost-effective ways of grouping multiple queries using algorithms designed to find common sub-expressions among these queries [136, 124]. Zhou el al. [157] present efficient and scalable techniques to find common sub-expressions among a batch of queries. Their solution chooses common sub-expressions which are computed only once, and the results are reused to answer other parts of queries. In [130, 70] the authors explore algorithms to provide cost-based optimizers that can maximize the shared processing of aggregate queries to speed up the real-time query processing.

The work presented in this dissertation (Chapter 5 integrates with any possible query optimizer regardless of the techniques used to combine or group multiple queries to produce one common query execution plan.

6.3.2 SECURITY AND PRIVACY PRESERVATION

Many data stream processing systems are increasingly being used to support applications that handle sensitive information, such as credit card numbers and healthcare records. These data must be protected from unauthorized access. Therefore, there is a need to integrate

access control mechanisms into data stream processing systems to achieve controlled and selective access to data streams. Though the data security community has a very rich history in developing access control models [32, 117], these models are largely tailored to traditional DBMSs. Thus, they cannot be readily adapted to data stream applications, mainly because: (a) traditional databases are static and bounded while data streams are unbounded and infinite; (b) queries over data streams are typically continuous and long-running unlike the one-time ad-hoc queries in traditional DBMSs; (c) in traditional DBMSs access control is enforced when users access the data, while in stream applications access control enforcement is data-driven (i.e., whenever data arrives). To cope with all these new requirements, there has been some ongoing research attempts to apply access control models in DSMSs. The research in this field can be divided into two different categories: cryptographic solutions and non-cryptographic solutions.

Carminati et al. provide access control via enforcing Role Based Access Control (RBAC) and secure operators [31, 30]. Operators are replaced with secure versions which determine whether a client can access a stream by referencing an RBAC policy. Their extended work allows interfacing with any stream processing system using query rewriting and middleware, as well as a wrapper to translate their queries into any language accepted by a DSMS.

Limiting disclosure in databases is studied in [93]. In data stream systems, limited disclosure is normally achieved by access control over streaming data. Lindner et al. [97] design a filtering operator and apply it to the stream query processing results to filter the output based on relevant access control policies. Nehme et al. [110] embed security policies into data stream, by security punctuations (SPs). Their query processor analyzes the SPs in each data stream, and enforces the policies during query processing. This framework is further improved by supporting dynamic access authorization of query issuers [109]. Ng et al. [111] also uses the principles of limited disclosure to limit who can access and operate on data streams, requiring queries to be rewritten to match the level at which they can access the data.

These systems are examples of non-cryptographic solutions but they require changing the underlying DSMS either by modifying the traditional operators to become security aware or by rewriting the queries, and therefore they are not globally applicable. Limited disclosure,

applies basic filtering to the query outputs which means that query operators execute at all times regardless of whether the output is used or not.

The solution proposed in this dissertation (Chapter 5 does not require any changes to the structure of the queries or the executing operators. The privacy switches placement algorithm 6 ensures an efficient execution of the query operators by minimizing the amount of unnecessary work of query operators for intermittent loss of access for users.

Streamforce [15] assumes an untrusted, honest-but-curious DDSMS and utilizes Attribute-Based Encryption (ABE) to enforce access control. The data providers will encrypt their data based on what attributes they desire a potential data consumer to possess. Streamforce is able to enforce access control over encrypted data through the use of their main access structure, views. The use of views in this system requires the data provider to be directly involved in the querying process and their work induces large decryption times of the data.

Similar cryptographic solutions can be found in CryptDB [119], PolyStream [140] and Sanctuary [139]. CryptDB allows computation over encrypted data on an untrusted honest-but-curious relational DBMS. CryptDBs primary goal is not access control, but rather allowing computation over encrypted data stored on an untrusted third-party database system. Similarly, PolyStream allows untrusted third-party infrastructure to compute on encrypted data, allowing in-network query processing and access control enforcement. PolyStream uses a combination of security punctuations, attribute-based encryption, and hybrid cryptography to enable flexible (ABAC) access control policy management and key distribution. Sanctuary alleviates the issues of the computation-enabling encryption techniques used so far that limits query expressiveness and can leak information about the underlying plaintext values. In doing so, Sanctuary uses Intels SGX as a trusted computing base for executing streaming operations on untrusted cloud providers.

Unfortunately, all the cryptographic and non-cryptographic solutions that have been proposed to maintain privacy through access control focused mainly on independently executing queries. All the previous work mentioned here did not consider applying access control measures on optimized multiple queries execution plans.

6.4 SUMMARY

The work presented in this dissertation has substantial differences to the previously proposed work in each of the three representative distributed systems. This dissertation addressed entirely unexplored spaces in each of these systems.

In Wireless Sensor Networks, no previously published work solves the problem of providing *both* confidentiality and fault tolerance for the process of in-network aggregation. The protocol proposed in this dissertation is novel in that it accomplishes both of these goals in an energy efficient manner for WSNs. It was shown that all the 3P requirements of this distributed system were achievable by adopting the appropriate algorithms and cryptographic techniques that are not prohibitively expensive.

In cloud servers, this dissertation highlighted for the first time in literature, the criticality of the cloud weak consistency models on access-control-based transactional databases. No other previous work addressed the confluence of data, policy, and credential inconsistency problems that can emerge as transactional database systems are deployed to the cloud. A wide range of solutions were presented in this dissertation that spans across all 3P properties. These wide range of solutions give the cloud users and system designers the flexibility to chose the best solution that fits their systems needs to achieve their desirable trade-off balance of the 3Ps.

Finally, in Data Stream Management Systems, no previous work discussed the conjunction of enforcing access control over shared operator networks. For the first time, this dissertation introduces a simple and efficient technique that allows DSMSs to apply access control policies over interleaved executing queries that are shared across multiple users. The proposed solution enforces the *need to share* and *need to protect* design models with high efficiency.

7.0 CONCLUSION

As the quality and ease of connectivity grows, and as the move into digital society becomes part of how this world operates, the dependency on distributed systems continues to increase. Speaking of stand-alone systems no longer makes any sense. The fact is simply that all systems developed are now connected one way or another. This dependency on distributed systems has increased the awareness and need that those systems can be justifiably relied upon: they appear to be doing what they are supposed to do. In many distributed systems the dependency on their correct behavior is evident until they break. Examples include those systems that provide services to users, such as cloud services, data query and processing systems, online stores and banking, and many more. These systems can appear that they are doing what they are supposed to be doing until some huge security or privacy breach shows up, or until they fail to provide accurate data to the users, or even when they stop providing their services in a timely way or they fail to operate within certain constraints.

It can be argued that a huge body of knowledge has been built regarding making distributed systems more dependable by providing better protection against deliberate attacks. Likewise, more research has focused on designing systems that maintain and process data while respecting the privacy and identity of users. And with the increased connectivity, much research has been spent on developing scalable solutions by which data can be easily attained. With this scalability and ease of data retrieval, data correctness and precision continues to be of a concern and some research work has focused on just that aspect of distributed systems development.

These elements, namely, user privacy, data precision and systems performance, which were referred to in this dissertation as the 3P properties, have been previously studied in isolation from one another. This dissertation brings in a new element in the distributed

systems research, namely viewing the 3P properties as a cohort of interleaved properties rather than disjoint properties. In other words, instead of concentrating only on the design of distributed systems that can achieve one or two of these properties simultaneously, much more emphasis is being put on viewing the 3Ps as a whole and finding the proper ways of achieving all of them in a balanced way.

There are considerable challenges with respect to achieving all 3P properties in distributed systems and the reason is that these properties compete against each other. Another reason is that these properties are not absolute properties. Given the diversity and heterogeneity of distributed systems, each system operates within different set of constraints and therefore the 3P requirements need to be defined and justified on a per system basis. Therefore, there is no such thing as a "one solution fits all". Accordingly, this dissertation focused on studying the interplay between the 3P properties in different representative classes of distributed systems. While only three representative applications were studied, one from each class of distributed systems, the lessons learned and conclusions made are invaluable results to the design of a wide range of distributed systems.

7.1 SUMMARY OF CONTRIBUTIONS

The majority of distributed systems research to date has focused on achieving one or two of the 3P properties at a time. As a result, the logical starting point for this dissertation was to ask, *"Is it possible to develop distributed systems that can balance among the three conflicting attributes: Privacy, Precision, and Performance (3Ps)?"*. A careful exploration of this question with respect to particular different classes of distributed systems was the subject of Chapters 3, 4, and 5, which led to the following contributions:

- *In Chapter 3*, the first class of distributed systems, WSNs, was presented. This class represented distributed systems that are characterized by high rates of communication failures due to the nature of their wireless connectivity, high computational and communication constraints due to the limited hardware capabilities, and high power consumption constraints. According to these set of constraints and characteristics of WSNs, the 3Ps

properties were carefully defined in the context of CWSNs applications. To satisfy the 3P properties in these set of applications, a protocol was proposed for reliably carrying out in-network aggregation (to achieve high data precision and high performance) while also maintaining the end-to-end privacy of individual sensor readings (for privacy). The proposed protocol uses a lightweight homomorphic cryptosystem to enable the in-network aggregation of encrypted values while imposing small computational overhead on individual sensors. Data precision was achieved through fault-tolerance techniques that make it possible for sensor readings that are lost due to link errors to be compensated for at most once. Also the proposed protocol provides exact aggregate results in case of no link failures rather than probabilistic query aggregate results. Simulations have shown that the proposed protocol achieves the desirable balance of the 3P properties by effectively providing end-to-end encryption and improvement of the root means square errors of the aggregate sensor readings while inducing minimal overheads on the power and bandwidth consumption of the WSN.

- *In Chapter 4*, the second class of distributed systems, cloud servers, was introduced. An example application of transactional database systems deployed over cloud servers was used as a case study for this class of distributed systems. Cloud servers have a very different set of constraints. They are characterized by their high availability and scalabililty constraints. In this chapter, the problems that result from the inconsistency of authorization policies used to govern data access in cloud servers were highlighted. The chapter presented the first formalization of the concept of trusted transactions when dealing with proofs of authorizations. To achieve the 3Ps balance, different levels of policy consistency constraints were defined along with different enforcement approaches to guarantee the trustworthiness of transactions executing on cloud servers. A Two-Phase Validation Commit (2PVC) protocol was proposed as a solution to enforce the different levels of consistencies before and during transaction commit times to ensure both data privacy and precision. A detailed system performance analysis was conducted through extensive simulations and trade-offs discussions of the 3Ps were presented to systems' designers to guide the decision making process of which consistency level best suits the different applications.

- *In Chapter 5*, the third distributed system, Data Stream Management Systems (DSMSs), was presented. DSMSs are characterized by their real-time data processing constraints. They deal with high input data rates and are required to process data in a fast efficient way. The problems of realizing all 3Ps were introduced in this system. The data privacy issues were presented using a motivating example. To achieve the desirable balance between all 3Ps, the chapter presented a simple and effective technique that allows DSMSs to interleave the execution of multiple overlapping queries that are shared by multiple users while applying access control restrictions on a per user or group of users. The proposed technique prevents DSMSs from fluctuating between different operator networks to execute in the different scenarios of access loss or gain (which in turn improves the response time and performance of the system). The solution proposed integrates new operators, Privacy Switches (PrS), into already existing shared operator networks. The PrSs are capable of seamlessly configuring the network to allow or deny access to certain query results complying with the access control policies defined in the system to ensure data privacy. Precision was guaranteed through the correct queries evaluations at all times despite the possible access control loss of some query results. Experimental evaluations proved that the proposed technique induces minimal overheads on the shared operator networks while achieving great gains in the network performance in the cases of intermittent access loss to some streams and queries. Accordingly, the 3Ps properties as defined by this system were successfully achieved in the balanced manner.

7.1.1 TOWARDS-3PS-BY-DESIGN

The dissertation's central premise is that distributed systems will become massively integrated in every aspect of the digital world, and in so doing generate significant need to think of the 3Ps as part of the design process of distributed systems rather than considering them as add-on properties. Not taking the interplay between the 3Ps during design time of distributed systems is what makes so many systems needlessly complex and results in flaws that need to be patched later on. This dissertation does not make the claim that the 3Ps properties were integrated into the initial design process of these distributed systems,

but rather were bolted on as add-on properties to fix the shortcomings of these systems in addressing the 3Ps properties in a balanced way.

While the goals were met and the systems were brought to a better integrated state of the 3Ps, the ultimate solution would have been to consider the 3Ps as an integral part of the initial designs of these systems. The result is that the 3Ps become an essential component of the core functionalities of distributed systems and hence the *3Ps-by-design* becomes the new design principle that is being promoted by this dissertation.

For many year, the well known Cavoukian's design principle of "privacy-by-design" [34] that was proposed in the early 90's was debated by many entities and governments. The privacy-by-design principle states that privacy must be incorporated into networked data systems and technologies, by default. Privacy must become integral to organizational priorities, project objectives, design processes, and planning operations and must be embedded into every standard, protocol and process.

While this design principle highlighted the need for incorporating privacy in the design of every system, this principle was consciously designed around the interests and needs of individual users, who have the greatest vested interest in the privacy of their own personal data. Yet, this design principle does not capture the problems that arise when systems engineers introduce privacy measures in their systems. Systems designers identified the need for data minimization as a necessary and foundational first step to engineer systems in line with the principles of privacy by design [72]. This means that systems should practice substantive data collection limitations as well as minimizing replication, retention and linkability of data to minimize the privacy risks and trust assumptions placed on other entities in the system.

It becomes very obvious now that privacy-by-design is a very narrow principle that is only concerned with users' privacy rather than many other important aspects of designing distributed systems, and that was one of the main reasons why this principle was critiqued for leaving many open questions about its application when engineering systems. Since the rise of this principle, many academic researchers and systems engineers have studied how to practice privacy-by-design from an engineering perspective in order to contribute to the closing of the gap between policy makers and engineers understanding of privacy-by-design [72].

This dissertation is pushing forward the need to re-think the privacy-by-design principle and is suggesting the use of the 3Ps-by-design as the a new and more comprehensive principle. 3Ps-by-design ensures users privacy is maintained without pushing for minimizing the need to share and replicate data. On the other hand, 3Ps-by-design encourages both the *need-to-share* and *need-to-protect* design principles to build scalable, reliable and collaborative distributed systems. Accordingly, this dissertation is introducing the following statement to define the the new notion of *3Ps-by-design*:

"Every distributed system should protect users privacy while ensuring the delivery of precise data without compromising systems' performance"

The case studies in this dissertation make evident that the application and balancing of 3Ps in distributed systems is an achievable goal, but one that requires that each system carefully crafts concrete definitions for each of the 3Ps that are best suited for its purpose. Most importantly, the three case studies underlined that the interpretation of the 3Ps properties requires specific contextual analysis, and an understanding of the multilateral systems' constraints, characteristics and properties.

The experiences gained by this dissertation when studying three different case studies of different distributed systems can be used to identify a number of general engineering principles that can be applied to provide some guidelines on how to achieve a balance between the 3Ps in distributed systems. Whether the 3Ps are being retrofitted into already existing distributed systems or introduced from the early design stages of new systems, there are common process elements that can be followed to ensure that these systems serve the purposes of balancing the 3Ps.

While these guidelines are not meant to provide a "3Ps-by-design checklist" that can be easily ticked away, they rather introduce the general theme of reasonable steps that can be used to approach a solution for balancing the 3Ps. The following are the general guidelines that this dissertation is proposing:

- *Identify and classify the system's entities*: This activity involves identifying the system components that are under the end user control, the system's control, or outside the control of both the end user and system. The interactions among the systems, entities,

and components need to be clearly stated, as do the constraints under which each entity operates. **Outcome**: A concrete and well defined system model and assumptions.

- *Define the 3Ps*: This involves clearly defining what is meant by privacy, precision and performance in the context of the specific system. Since these are not absolute properties, their definitions have to be clearly stated and justified with respect to the systems requirements, assumptions and constraints. **Outcome**: A concrete definition of the 3Ps.
- *Model attackers, threats, and risks*: Start developing models of potential attackers and the types of threats these attackers could realize. The security and trust assumptions of each entity in the system should be carefully identified. The likelihood and impact of the realization of the threats are then used in the risk analysis. This is not a trivial exercise, and requires analytical expertise as well as awareness of recent research results on potential attacks and vulnerabilities. **Outcome**: A risk analysis model.
- *Articulate the success criteria*: In this step, the criteria by which the systems' designers can measure the effectiveness of their solutions for balancing the 3Ps need to be identified. This involves quantifying the 3Ps, if possible. **Outcome**: A set of test cases to conduct once the design is ready.
- *Select the tools and protocols to use*: The right set of protocols, algorithms and tools are to be carefully chosen to achieve both privacy and precision within the systems' identified constraints. The impact of the chosen algorithms and protocols on the system's performance need to be analyzed with every choice made. **Outcome**: A system design model.
- *Implementation and testing of the design*: The final step is to implement the solutions that fulfills the 3Ps requirements. Further, the potential vulnerabilities have to be examined, and the functioning of the system according to the articulated success criteria have to be validated. **Outcome**: A 3Ps compliant system.

Following these guidelines, systems designers will be able to give adequate attention in studying the trade-offs of the 3Ps throughout the development lifecycles of distributed systems. Accordingly, the *3Ps-by-design* principle will allow systems to be made more user-centric by respecting users' privacy and considering it as an integral part of systems design while still being able to deliver high precision data without sacrificing systems performance.

This will provide support for balancing privacy, precision and performance in distributed systems from its very own foundations.

7.2 FUTURE WORK

In this dissertation, it was shown that, with the right set of protocols and algorithms, the desirable 3P properties can co-exist in a balanced way in distributed systems. The approaches presented in this dissertation in each representative system are examples of how the 3Ps can be achieved in each class of distributed system but they are not constrained to those systems. These approaches can be generalized and modified to suit a broader range of distributed systems that carry similar attributes or constraints. For example, the secure in-network aggregation protocol presented for WSNs can be generalized to execute on similar systems such as distributed control systems, smart homes, collaborative crowd sourcing applications, and IoT applications. Similarly, the consistency approaches presented in the cloud databases system can be generalized and used to ensure both policy and data consistencies in any policy-based access control distributed system.

Therefore, this dissertation leaves the door open for more work to be done along the same lines. It is worth the investigation to find adaptations to the proposed protocols and algorithms presented in this dissertation to make them suit the constraints and attributes of other distributed systems.

Another important area to investigate is the interesting aspect of this dissertation that showed the deriving need to increase the focus on the new design principle "3Ps-by-design" in designing and developing distributed systems. This research has shown that designing systems that balance all 3P properties is possible and would successfully lead to more secure and reliable distributed systems without the potential side effects of compromising performance.

As indicated earlier in this dissertation, there are several mission-critical distributed systems that demonstrate the need for incorporating the 3Ps in their design. According to the CISA (Cybersecurity and Infrastructure Security Agency) [4], there are 16 critical infrastructure sectors that compose the assets, systems, and networks, whether physical or

virtual, so vital to the United States that their incapacitation or destruction would have a debilitating effect on security, national economic security, national public health or safety, or any combination thereof. Examples of these sectors are healthcare and public health, critical manufacturing, emergency services, financial, oil and gas industry, etc. Reexamining the design of systems within these sectors with the new "3Ps-by-design" principal in place would result in the evolution of more trutworthy, reliable and efficient systems and potentially a plethora of research opportunities.

The next natural step to follow up on the work presented in this dissertation would be to explore trends in distributed systems design and implementation that can be used to enforce the "3Ps-by-Design" principle. In other words, finding the proper formalism for describing the observed behavior required by distributed systems towards the 3Ps-by-design. Exploring these trends will have far-reaching consequences in the future of distributed systems development.

While the "3Ps-by-design" concept seems to be an interesting design concept, formalizing the foundational principles of this concept and promoting it to be the new standard design principle of distributed systems is a challenging task. It took years of negotiations and debates to have the privacy-by-design principle promote from being just an idea to be widely adopted by the community of engineers and systems designers. It is anticipated that a more stringent, yet more comprehensive principle as the 3Ps-by-design to face similar challenges. Yet, this dissertation is hoping to bring this new design concept to light and get more researchers and systems designers to believe in it and adopt it.

BIBLIOGRAPHY

[1] Amazon Elastic Compute Cloud (Amazon EC2). https://aws.amazon.com/ec2/.

[2] Amazon Simple Storage Service (Amazon S3) . https://aws.amazon.com/s3/.

[3] Apple Health. https://www.apple.com/lae/ios/health/.

[4] Cybersecurity and Infrastructure Security Agency. https://https://www.cisa.gov/.

[5] Googe App Engine. https://cloud.google.com/appengine/.

[6] Google Fit. https://www.google.com/fit/.

[7] IBM Streaming Analytics. https://www.ibm.com/cloud/streaming-analytics.

[8] Microsoft Azure Stream Analytics. https://azure.microsoft.com/en-us/services/stream-analytics/.

[9] Oracle Stream Analytics. https://www.oracle.com/middleware/technologies/stream-processing.html.

[10] Samsung S-Health. https://www.samsung.com/us/samsung-health/.

[11] Daniel Abadi, Wolfgang Lindner, Samuel Madden, and Jrg Schuler. An integration framework for sensor networks and data stream management systems. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases(VLDB*, pages 1361–1364, 12 2004.

[12] Daniel J. Abadi. Data management in the cloud: Limitations and opportunities. IEEE Data Engineering Bulletin, Mar. 2009.

[13] Daniel J. Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, pages 120–139, August 2003.

[14] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, pages 393 – 422, 2002.

[15] Dinh Tien Tuan Anh and Anwitaman Datta. Streamforce: Outsourcing access control enforcement for stream data to the clouds. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY, pages 13–24. ACM, 2014.

[16] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. Stream: The stanford stream data manager. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pages 665–665. ACM, 2003.

[17] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*, pages 121–142, June 2006.

[18] Michael Armbrust et al. Above the clouds: A berkeley view of cloud computing. Technical report, University of California, Berkeley, Feb. 2009.

[19] Frederik Armknecht, Dirk Westhoff, Joao Girao, and Alban Hessler. A lifetime-optimized end-to-end encryption scheme for sensor networks allowing in-network processing. *Computer Communications*, pages 734–749, Mar 2008.

[20] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609. ACM, 2007.

[21] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 598–609. ACM, 2007.

[22] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, pages 2787–2805, Oct 2010.

[23] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16. ACM, 2002.

[24] Andreea Berfield, Panos K. Chrysanthis, and Alexandros Labrinidis. Efficient handling of sensor failures. In *Proceedings of the 3rd Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2006*, DMSN '06, pages 33–40. ACM, 2006.

[25] Andreea Berfield, Panos K. Chrysanthis, and Daniel Mossè. Lsynd: Localized synopsis diffusion. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 313–320, May 2007.

[26] Sanjit Biswas and Robert Morris. Exor: Opportunistic multi-hop routing for wireless networks. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '05*, pages 133–144. ACM, 2005.

[27] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 187–198. ACM, 2009.

[28] Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC. ACM, 2000.

[29] Frank J Cangialosi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, Jan 2005.

[30] Barbara Carminati, Elena Ferrari, and Kian Lee Tan. Enforcing access control over data streams. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT, pages 21–30. ACM, 2007.

[31] Barbara Carminati, Elena Ferrari, and Kian Lee Tan. Specifying access control policies on data streams. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, DASFAA, pages 410–421. Springer-Verlag, 2007.

[32] Silvana Castano, Maria G. Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. Addison-Wesley, 1995.

[33] Claude Castelluccia, Aldar C-F. Chan, Einar Mykletun, and Gene Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Transactions on Sensor Networks*, pages 20:1–20:36, Jun 2009.

[34] Ann Cavoukian. Privacy by design: The 7 foundational principles. In *Information and Privacy Commissioner of Ontario*, 2009.

[35] Prabhakar Chaganti and Rich Helms. *Amazon SimpleDB Developer Guide*. Packt Publishing, 1st edition, 2010.

[36] Haowen Chan, Virgil D. Gligor, Adrian Perrig, and Gautam Muralidharan. On the distribution and revocation of cryptographic keys in sensor networks. *IEEE Trans. Dependable Secur. Comput.*, pages 233–247, July 2005.

[37] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP. IEEE Computer Society, 2003.

[38] Haowen Chan, Adrian Perrig, and Dawn Song. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM conference on Computer and communications security, CCS '06*, pages 278–287. ACM, 2006.

[39] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pages 668–668. ACM, 2003.

[40] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, pages 4:1–4:26, Jun 2008.

[41] P. Chatterjee, L. J. Cymberknop, and R. L. Armentano. Iot-based decision support system for intelligent healthcare applied to cardiovascular diseases. In *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 362–366, Nov 2017.

[42] Chung-Min Chen, H. Agrawal, M. Cochinwala, and D. Rosenbluth. Stream query processing for healthcare bio-sensor applications. In *Proceedings. 20th International Conference on Data Engineering*, pages 791–794, April 2004.

[43] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 379–390. ACM, 2000.

[44] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Uur etintemel, Ying Xing, and Stan Zdonik. Scalable distributed stream processing. In *In CIDR*, 2003.

[45] P. K. Chrysanthis, G. Samaras, and Y. J. Al-Houmaily. Recovery and performance of atomic commit processing in distributed database systems. In *Recovery Mechanisms in Database Systems*. PHPTR, 1998.

[46] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, Mar 2004.

[47] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, pages 1277–1288, Aug 2008.

[48] George F Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed systems: concepts and design*. Harlow, England : Addison-Wesley, 5th edition, 2012.

[49] Ivan Damgaard, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious ram without random oracles. In Yuval Ishai, editor, *Theory of Cryptography*, pages 144–163. Springer Berlin Heidelberg, 2011.

[50] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Elastras: an elastic transactional data store in the cloud. In *USENIX HotCloud*, 2009.

[51] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Operating Systems Review*, pages 205–220, Oct 2007.

[52] T. Dillon, C. Wu, and E. Chang. Cloud computing: Issues and challenges. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, Apr 2010.

[53] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram Khalili. A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions on Information Systems Security*, pages 228–258, May 2005.

[54] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson, 7th edition, 2015.

[55] Sinem Coleri Ergen and Pravin Varaiya. Tdma scheduling algorithms for wireless sensor networks. *Wireless Networks*, pages 985–997, May 2010.

[56] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: A survey. *Wireless Communications, IEEE*, pages 70–87, Apr 2007.

[57] Stefan Felsner, Giuseppe Liotta, and Stephen K. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. In *Revised Papers from the 9th International Symposium on Graph Drawing, GD '01*, pages 328–342. Springer-Verlag, 2002.

[58] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, pages 34–64, Feb 1999.

[59] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, pages 182–209, Sep 1985.

[60] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, pages 11–25, Oct 2001.

[61] Minos Garofalakis. Proof sketches: Verifiable in-network aggregation. In *IEEE Internation Conference on Data Engineering, ICDE'07*, pages 996–1005, Apr 2007.

[62] Sandra Geisler. Data stream management systems. In *Data Exchange, Information, and Streams*, 2013.

[63] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, pages 51–59, June 2002.

[64] Sameh Gobriel, Sherif Khattab, Daniel Mossè, Josè Brustoloni, and Rami Melhem. Ridesharing: Fault tolerant aggregation in sensor networks using corrective actions. In *3rd Annual IEEE Communications Society on Sensor, Mesh and Ad Hoc Communications and Networks, SECON'06*, pages 595–604, Sep 2006.

[65] Lukasz Golab and Tamer Ozsu M. Issues in data stream management. In *SIGMOD Record*, pages 5–14. ACM, 2003.

[66] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43(3):431–473, May 1996.

[67] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[68] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 157–167. Society for Industrial and Applied Mathematics, 2012.

[69] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 1st edition, 1992.

[70] Shenoda Guirguis, Mohamed A. Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. Optimized processing of multiple aggregate continuous queries. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM, pages 1515–1524. ACM, 2011.

[71] Hongfei Guo, PerAAke Larson, Raghu Ramakrishnan, and Jonathan Goldstein. Relaxed currency and consistency: How to say "good enough" in sql. In *SIGMOD International Conference on Management of Data*, pages 815–826. ACM, 2004.

[72] Seda F. Gürses, Carmela Troncoso, and Claudia Díaz. Engineering privacy by design. 2011.

[73] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 270–283. ACM, 2004.

[74] Wenbo He, Xue Liu, Hoang Nguyen, K. Nahrstedt, and T.T. Abdelzaher. Pda: Privacy-preserving data aggregation in wireless sensor networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications*, pages 2045 –2053, May 2007.

[75] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, Jan 2000.

[76] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[77] G. Hoblos, M. Staroswiecki, and A. Aitouche. Optimal design of fault tolerant sensor networks. In *Proceedings of the 2000 IEEE International Conference on Control Applications.*, pages 467–472, Sep 2000.

[78] Vincent C. Hu, Richard Kuhn, David F. Ferraiolo, and Jeffery Voas. Attribute-based access control. *Computer*, 48:85–88, Feb 2015.

[79] Marian K. Iskander, Adam J. Lee, and Daniel Mossè. Privacy and robustness for data aggregation in wireless sensor networks. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS, pages 699–701. ACM, 2010.

[80] Marian K. Iskander, Adam J. Lee, and Daniel Mossè. Confidentiality-preserving and fault-tolerant in-network aggregation for collaborative wsns. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 107–116, Oct 2012.

[81] Marian K. Iskander, Tucker Trainor, Dave W. Wilkinson, Adam J. Lee, and Panos K. Chrysanthis. Balancing performance, accuracy, and precision for secure cloud transactions. *IEEE Trans. Parallel Distrib. Syst.*, pages 417–426, Feb 2014.

[82] Marian K. Iskander, Dave W. Wilkinson, Adam J. Lee, and Panos K. Chrysanthis. Enforcing policy and data consistency of cloud transactions. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops*, ICDCSW, pages 253–262. IEEE Computer Society, 2011.

[83] G. Kakamanshadi, S. Gupta, and S. Singh. A survey on fault tolerance techniques in wireless sensor networks. In *International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 168–173. IEEE, Oct 2015.

[84] Jong Wook Kim, Beakcheol Jang, and Hoon Yoo. Privacy-preserving aggregation of personal health data streams. *PLOS ONE*, 13:1–15, 11 2018.

[85] Tim Kraska, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. Consistency rationing in the cloud: Pay only when it matters. *Proc. VLDB Endow.*, pages 253–264, Aug 2009.

[86] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication. In *RFC 2104*. RFC Editor, Feb. 1997.

[87] C. P. Kruger and G. P. Hancke. Implementing the internet of things vision in industrial wireless sensor networks. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 627–632, July 2014.

[88] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Operating Systems Review*, pages 35–40, Apr 2010.

[89] Adam J. Lee, Kazuhiro Minami, and Marianne Winslett. Lightweight consistency enforcement schemes for distributed proofs with hidden subtrees. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, pages 101–110, New York, NY, USA, 2007. ACM.

[90] Adam J. Lee, Kazuhiro Minami, and Marianne Winslett. On the consistency of distributed proofs with hidden subtrees. *ACM Transactions on Information and Systems Security*, pages 25:1–25:32, July 2010.

[91] Adam J. Lee and Marianne Winslett. Safety and consistency in policy-based authorization systems. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 124–133. ACM, 2006.

[92] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431 – 440, 2015.

[93] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB, pages 108–119. VLDB Endowment, 2004.

[94] Tiancheng Li and Ninghui Li. On the tradeoff between privacy and utility in data publishing. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 517–526. ACM, 2009.

[95] Wenfeng Li, Junrong Bao, and Weiming Shen. Collaborative wireless sensor networks: A survey. In *International Conference on Systems, Man, and Cybernetics*, pages 2614–2619. IEEE, Oct 2011.

[96] Wang Licheng, Wang Lihua, Pan Yun, Zhang Zonghua, and Yang Yixian. Discrete logarithm based additively homomorphic encryption and secure data aggregation. *Inf. Sci.*, pages 3308–3322, August 2011.

[97] W. Lindner and J. Meier. Securing the borealis data stream engine. In *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, pages 137–147, Dec 2006.

[98] Donggang Liu and Peng Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communications security, CCS '03*, pages 52–61, 2003.

[99] Q. Ma, J. Zhang, Y. Peng, W. Zhang, and D. Qiao. Se-oram: A storage-efficient oblivious ram for privacy-preserving access to cloud storage. In *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 20–25, June 2016.

[100] S. Madden and M. J. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. In *Proceedings 18th International Conference on Data Engineering*, pages 555–566, Feb 2002.

[101] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review*, pages 131–146, Dec 2002.

[102] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 49–60. ACM, 2002.

[103] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, 2011.

[104] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research, CIDR*, pages 245–256, 2003.

[105] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - ocsp. RFC 2560, June 1999. http://tools.ietf.org/html/rfc5280.

[106] E. Mykletun, J. Girao, and D. Westhoff. Public key based cryptoschemes for data concealment in wireless sensor networks. In *In IEEE International Conference on Communications, ICC'06*, volume 5, pages 2288–2295, Jun 2006.

[107] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Transactions on Storage (TOS)*, pages 107–138, May 2006.

[108] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd*

international conference on Embedded networked sensor systems, SenSys '04, pages 250–262. ACM, 2004.

[109] Rimma V. Nehme, Hyo-Sang Lim, and Elisa Bertino. Fence: Continuous access control enforcement in dynamic data stream environments. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY, pages 243–254. ACM, 2013.

[110] Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. A security punctuation framework for enforcing access control on streaming data. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE, pages 406–415. IEEE Computer Society, 2008.

[111] Wee S. Ng, Huayu Wu, Wei Wu, Shili Xiang, and Kian-Lee Tan. Privacy preservation in streaming data collection. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 810–815, Dec 2012.

[112] P. Pace, G. Aloi, G. Caliciuri, R. Gravina, C. Savaglio, G. Fortino, G. Ibanez-Sanchez, A. Fides-Valero, J. Bayo-Monton, M. Uberti, M. Corona, L. Bernini, M. Gulino, A. Costa, I. De Luca, and M. Mortara. Inter-health: An interoperable iot solution for active and assisted living healthcare services. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 81–86, April 2019.

[113] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology, EUROCRYPT'99*, pages 223–238. Springer-Verlag, 1999.

[114] J Park and A Segev. Using common subexpressions to optimize multiple queries. In *In Proceedings of 4th Internation Conference on Data Engineering*, pages 311–319. IEEE, 1988.

[115] N. R. Patel and S. Kumar. Wireless sensor networks challenges and future prospects. In *International Conference on System Modeling Advancement in Research Trends (SMART)*, pages 60–65. IEEE, Nov 2018.

[116] Steffen Peter, Krzysztof Piotrowski, and Peter Langendoerfer. On concealed data aggregation for wireless sensor networks. In *Proceedings of the 24th International conference of the Computer and Communications Societies*. IEEE, 2005.

[117] Mario Piattini and Oscar Diaz, editors. *Advanced Database Technology and Design*. Artech House, Inc., 1st edition, 2000.

[118] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. In *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 404–413, May 1995.

[119] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings*

of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP, pages 85–100. ACM, 2011.

[120] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212. ACM, 2009.

[121] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, Feb 1978.

[122] Rodrigo Roman, Cristina Alcaraz, and Javier Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Networks and Applications*, pages 231–244, Aug 2007.

[123] Arnon Rosenthal and Sharma Chakravarthy. Anatomy of a mudular multiple query optimizer. In *In Proceedings of the 14th International Conference on Very Large Data Bases*, pages 230–239, 1988.

[124] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 249–260. ACM, 2000.

[125] R. S. Sandhu and P. Samarati. Access control: Principle and practice. *IEEE Communications Magazine*, 32(9):40–48, Sep. 1994.

[126] C. Schurgers and M. B. Srivastava. Energy efficient routing in wireless sensor networks. In *MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force*, pages 357–361, Oct 2001.

[127] F. Seb, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J. Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Transactions on Knowledge and Data Engineering*, pages 1034–1038, Aug 2008.

[128] Timos K Sellis. Multiple-query optimization. In *ACM Transactions on Database Systems*, pages 23–52. ACM, 1988.

[129] Mohamed A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, and Panos K. Chrysanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access, MobiDe '03*, pages 69–76. ACM, 2003.

[130] Anatoli U. Shein, Panos K. Chrysanthis, and Alexandros Labrinidis. F1: Accelerating the optimization of aggregate continuous queries. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM, pages 1151–1160. ACM, 2015.

[131] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung. Recent advances in industrial wireless sensor networks toward efficient management in iot. *IEEE Access*, pages 622–637, 2015.

[132] Kyuseok Shim, Timos Sellis, and Dana Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data Knowledge Engineering*, pages 197–222, 1994.

[133] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys '04*, pages 188–200. ACM, 2004.

[134] Alexander Shraer, Christian Cachin, Asaf Cidon, Idit Keidar, Yan Michalevsky, and Dani Shaket. Venus: Verification for untrusted cloud storage. In *Proceedings of the 2nd ACM Cloud Computing Security Workshop*, pages 19–30, 2010.

[135] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys '04*, pages 239–249. ACM, 2004.

[136] Subbu N. Subramanian and Shivakumar Venkataraman. Cost-based optimization of decision support queries using transient-views. *SIGMOD Rec.*, pages 319–330, 1998.

[137] Andrew S Tanenbaum and Maarten van Steen. *Distributed systems : principles and paradigms*. Upper Saddle RIiver, NJ : Pearson Prentice Hall, 2nd edition, 2007.

[138] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 309–324. ACM, 2013.

[139] Cory Thoma, Adam Lee, and Alexandros Labrinidis. Behind enemy lines: Exploring trusted data stream processing on untrusted systems. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, CODASPY, pages 243–254. ACM, 2019.

[140] Cory Thoma, Adam J. Lee, and Alexandros Labrinidis. Polystream: Cryptographically enforced access controls for outsourced data stream processing. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, SACMAT, pages 227–238. ACM, 2016.

[141] Maarten Van Steen and Guillaume Pierre. *Replicating for Performance: Case Studies*, pages 73–89. Springer Berlin Heidelberg, 2010.

[142] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, pages 45–52, Jan 2011.

[143] Werner Vogels. Eventually consistent. *Queue*, 6(6):14–19, Oct 2008.

[144] Werner Vogels. Eventually consistent. *Communications of the ACM*, pages 40–44, Jan 2009.

[145] Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR*, pages 134–143, 01 2011.

[146] X. Wang, S. Yang, S. Wang, X. Niu, and J. Xu. An application-based adaptive replica consistency for cloud storage. In *2010 Ninth International Conference on Grid and Cloud Computing*, pages 13–17, Nov 2010.

[147] Zhou Wei, Guillaume Pierre, and Chi-Hung Chi. Cloudtps: Scalable transactions for web applications in the cloud. *IEEE Transactions on Services Computing*, pages 525–539, Apr 2012.

[148] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pages 139–148. ACM, 2008.

[149] Peter Williams, Radu Sion, and Dennis Shasha. The blind stone tablet: Outsourcing durability to untrusted parties. In *Proceedings of the Network and Distributed System Security Symposium*, 2009.

[150] Ted Wobber, Thomas L. Rodeheffer, and Douglas B. Terry. Policy-based access control for weakly consistent replication. In *Proceedings of the 5th European Conference on Computer Systems*, pages 293–306. ACM, 2010.

[151] L. Xiang, J. Luo, and A. Vasilakos. Compressed data aggregation for energy efficient wireless sensor networks. In *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 46–54, June 2011.

[152] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. Sdap: A secure hop-by-hop data aggregation protocol for sensor networks. *ACM Transactions on Information and System Security*, pages 18:1–18:43, Jul 2008.

[153] Wei Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1567–1576, June 2002.

[154] Weihai Yu, Yan Wang, and Calton Pu. A dynamic two-phase commit protocol for self-adapting services. In *IEEE SCC*, 2004.

[155] Ke Zeng. Publicly verifiable remote data integrity. In *Proceedings of the 10th International Conference on Information and Communications Security*, ICICS '08, pages 419–434. Springer-Verlag, 2008.

[156] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems, SenSys '03*, pages 1–13. ACM, 2003.

[157] Jingren Zhou, Per-Ake Larson, Johann-Christoph Freytag, and Wolfgang Lehner. Efficient exploitation of similar subexpressions for query processing. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 533–544. ACM, 2007.

[158] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin. Iot gateway: Bridging wireless sensor networks into internet of things. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 347–352, Dec 2010.