Modern Applied Science; Vol. 12, No. 5; 2018
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

PPUSTMAN: Privacy-Aware PUblish/Subscribe IoT MVC Architecture Using Information Centric Networking

Huda K. Saadeh[1,2], Wesam Almobaideen[1] & Khair Eddin Sabri[1]

[1] Faculty of Information Technology, The University of Jordan, Amman, Jordan

[2] Faculty of Information Technology, University of Petra, Amman, Jordan

Correspondence: Huda K. Saadeh, Faculty of Information Technology, The University of Jordan, Amman, Jordan. Tel: 962-799-703-777. E-mail: hsaadeh@uop.edu.jo

Received: March 22, 2018 Accepted: April 25, 2018 Online Published: April 30, 2018

doi:10.5539/mas.v12n5p128 URL: https://doi.org/10.5539/mas.v12n5p128

Abstract

IoT applications have been evolved in relation to every aspect of human life. The design of the applications adds new challenges such as mobility, scalability, and privacy to the current networking architecture design. For that reason, it is mandatory to investigate new solutions and paradigms such as Information Centric Network (ICN). ICN handles many of the challenges barely handled by current IP networks such as mobility and scalability, by using in-network caching and content retrieving based on contents names instead of host addresses. This paper presents a privacy-aware ICN architecture for IoT environments based on the Model-View-Controller (MVC) publish/Subscribe communication approach. This architecture focuses on supporting mobility, scalability, and privacy. A communication and processing cost comparison between Publish/Subscribe N-Tier and Publish/Subscribe MVC architectures, shows that the later outperforms N-Tier in communication, processing cost, and parallelism capabilities. Reasoning and planning for publishing actuation commands scenarios is performed using Situation Calculus which is used to formalize the communications in causality and temporal framework.

Keywords: Information-centric network (ICN), Internet of Things (IoT), Model-View-Controller (MVC), Publish/Subscribe Architecture, Situation Calculus

1. Introduction

IoT adds new challenges such as mobility, scalability and security over current networking paradigm (Atzori, et al., 2010), (Lindgren, et al., 2014), (Almobaideen, et al., 2017). Standardization organization such as ITEF, IEEE, and researchers around the world have been developing and testing new protocol standards that alleviate the problems. Among these efforts is proposing generic five layers stack that suit IoT constrained devices (Kraijak & Tuwanut, 2015), (Saadeh, et al., 2017), and proposing different layer's lightweight protocols for embedded and constrained devices such as CoAP, MQTT, and 6LoWPAN (Solapure & Kenchannavar, 2016), (SHENG, et al., 2013). Unfortunately these solutions barely covered the symptoms, and raised the need for new Internet paradigms (Li, et al., 2015) (Li, et al., 2014) (Atzori, et al., 2010), (Amadeo, et al., 2016). Future Internet networking paradigms such as Information Centric Network (ICN) has been one of the hottest research topics in the literature, due to the need for effective solutions to treat current network problems and handle new challenges (Atzori, et al., 2010), (Lindgren, et al., 2014), (Li, et al., 2014).

ICN paradigm is taking the attention of researchers for its capability to handle many of IoT challenges such as mobility, energy consumption, resources constrained devices, communications patterns, and network traffic load (Ahlgren, et al., 2012), (Amadeo, et al., 2016), (Xylomenos, et al., 2014). ICN shows better challenges handling comparing to current host-centric network paradigm, by using in-network and application-independent caching for any retrieved contents, and location-independent names to identify services and contents (Lindgren, et al., 2014), (Quevedo, et al., 2014). ICN in-network caching and naming schemes support mobility of IoT devices, where user's requests are served by the nearest cached copy without bothering of content's provenance location (Li et al., 2014).

Contents are considered as Named Data Objects (NDOs) in ICN and are named using different naming schemes. Naming schemes in ICN are either flat or hierarchical (Bari, et al., 2012). The former is cryptographic fixed size,

usually of the form <P: L> where P is the owner or provider public key hash, and L is the content's hash or unique label identified by the content's owner or provider (Xylomenos, et al., 2014). The later is similar to URLs, variable length, human readable names, and allows aggregations. While flat names could be self-certifying, where hash of provider's public key and content's hash are included as part of the name to verify contents authenticity and integrity, hierarchical names are human readable and more appropriate for dynamic data which will be available in future. Self-certifying names provide content-name binding, meanwhile hierarchical names provide name-provenance binding (Ghodsi, et al., 2011), (Koponen, et al., 2007).

Content's discovery in ICN is done either by using Name Based Routing (NBR) or Name Resolution Systems (NRS) mechanisms (Bari, et al., 2012), (Xiaoke, et al., 2015). The former routes user's requests hop-by-hop using names rather than IPs. It matches names in request packets with forwarding information in internal forwarding tables. The later, NRS, usually uses centralized systems that keep information about registered names, map them to their locators, and serve requests by the most suitable content sources according to network condition and servers load (Bari, et al., 2012), (Fotiou, et al., 2010). Caching is a core characteristic in ICN, where caching policies and placement are very important to be considered while designing any ICN architecture (Ioannou & Weber, 2016), (Jin, et al., 2017). Caching should be implemented for all applications' packets, and for all users' requests. This is a major difference between ICN and Content Defined Network (CDN) (Amadeo, et al., 2016), (Suarez, et al., 2016).

Among the most important challenges in designing ICN-IoT architectures is the consideration of IoT communication patterns. Pull-based communications in IoT load the network with periodic or event data requests which can be delivered more efficiently using Publish/Subscribe (Pub/Sub) approach (Happ, et al., 2017). Architecture design process includes steps of selecting architectural styles that meet and handle pre-defined goals and constraints. Network architectural styles and other software architectural styles have been described by Fielding in (Fielding, 2000), (Fielding & Taylor, 2002) such as Client-Server, structured and unstructured Peer-to-Peer, m-Tier, Event-Based (Pub/Sub), and Model-View-Controller (MVC). Moreover, he has discussed each style impact, advantages, and disadvantages on architectures performance according to design goals and constraints.

This paper proposes privacy-aware architecture based on ICN paradigm to support IoT systems development, using Pub/Sub and MVC architectural styles. MVC design pattern model supports scalability, maintainability (GuangChun, et al., 2003), and reduces complexity of architecture design process (Kim, et al., 2015). Pub/Sub decouples publishers and subscribers in space and time, which supports mobility, privacy, and efficient delivery of IoT data (Happ, et al., 2017). The constraints that this architecture should consider are mobility, scalability, interoperability, and privacy.

This paper is organized as the following: Section 2 summarizes literature in ICN-IoT field. The proposed architecture is explained in details in Section 3. Section 4 analyzes and discusses the communication and processing cost of N-Tier, MVC architectures, Publish/Subscribe and ICN caching models. Moreover, reasoning and planning authenticated actuation scenarios are formalized using situation calculus in Section 4. Finally, Section 5 concludes the paper and presents future work.

2. Literature Review

This section presents an overview of proposed ICN architectures in the literature, and privacy issues considered in some of them.

2.1 ICN Architectures Structure

Proposed architectures for ICN field have been increased recently. DONA (Koponen, et al., 2007), NDN (Jacobson, et al., 2009), PSIRP (Fotiou, et al., 2010), NetInf (Dannewitz, et al., 2013) are among the earlier proposals which concentrate on functionalities such as data naming, caching and routing, that are known as the data plane. NetInf and PSIRP consider additional functionalities such as Pub/Sub approach and monitoring the network for network-aware routing, which can be considered as initial steps of adding control functionalities to the architectures. Later ICN architectures such as COMET and CONVERGENCE have multiple planes architecture and different components.

First proposals in ICN paradigm start before taking IoT into consideration. Lately, many ICN architecture research trends were moving toward using ICN for IoT such as MobilityFirst (Seskar, et al., 2011), GreenICN (TAGAMI & ARUMAITHURAI, 2016), and C-DAX (Hoefling, et al., 2015). MobilityFirst considers fast discovery and routing of data in highly mobile environments. Authors of (Chen, et al., 2016) adapted MobilityFirst architecture for IoT using data discovery and routing in IoT scenarios. GreenICN is designed for

emergency and crisis situations where both pub/sub and pull-based approaches where adopted to deliver data for users located in disjoint networks. C-DAX is designed as a secure architecture for smart grid communication. These last three architectures have control-aware functionalities to improve network performance. Information Centric Networking Research Group (ICNRG) from Internet Research Task Force (IRTF) has proposed an ICN-IoT middleware (Sicari, et al., 2017) that supports IoT requirements and handles some secured communications to deliver IoT information between various parties. The proposed middleware (Sicari, et al., 2017) separates between IoT tasks and ICN tasks. ICN plane is responsible for naming, data delivery and caching through the ICN-IoT server. In IoT plane, aggregators collect sensors data and communicate with other aggregators and gateways. Local Service Gateways (LSG) controls the communication between IoT local system and global IoT systems (Sicari, et al., 2017).

CONVERGENCE and COMET separate data plane from control plane, by having additional components or parties with specific functionalities that differ from routing and caching contents. Data plane in CONVERGENCE is built using CoNet (Detti, et al., 2011) which is an ICN approach to cache and deliver data. CONVERGENCE consists of three layers: the CONVERGENCE Application layer (CoApp), CONVERGENCE Middleware layer (CoMid), and CONVEGENCE Computing Platform layer (CoComp). The CoComp consists of CONVERGENCE Network (CoNet) and CONVERGENCE Security (CoSec). CoMid is considered as the control plane which controls contents creation, and consumption through pub/sub approach. It also orchestrates set of technology engines activated by the application layer to abstract interaction of applications and the CoComp layer. CoMid interacts with CoNet and CoSec using CoNet and CoSec APIs, as well, as it interacts with CoApp using CoMid APIs (Detti, et al., 2011), (Salsano, et al., 2011), (Melazzi, 2011).

COMET data plane is built using CURLENG ICN architecture (Chai, et al., 2011). The control plane in COMET, known as Content Mediation Plane (CMP), is separated from the Content Forwarding Plane (CFP). CMP includes a Content Resolution Function (CRF) node which is responsible of name resolution, Content Mediation Function (CMF) node which receives information about the network and content's servers from the Server and Network Monitoring Function (SNMF) node and cooperates with the Path Management Function (PMF) node to determine the best path to route the information. Routing in COMET is the responsibility of the Content-Aware Forwarding Function (CAFF) nodes in CFP which forward the information using the path defined by the CMF (Pavlou, et al., 2013).

C-DAX as mentioned above has been designed as secured pub/sub IoT-ICN architecture for smart grid applications. C-DAX differentiated data plane communications from control plane communications. Control plane communications include publishers and subscribers interactions with security and resolution servers. Meanwhile data plane communications include the data forwarding messages and mechanisms from data sources, known as Data Brokers (DB), to the authorized subscribers. Data in C-DAX is organized as topic data, stored in Data Brokers (DBs), and a topic-DB mapping can be resolved by a Resolver Server (RS). Control functionality in C-DAX is implemented in cloud including the RS and the Security Server (SecServ). C-DAX client's interaction with C-DAX cloud is implemented through Designated Nodes (DN) which also forward the topic data to/from the cloud (Hoefling, et al., 2015), (Heimgaertner, et al., 2015), (Hoefling, et al., 2016).

While C-DAX DNs can be implicitly considered as edge computing, in (Amadeo, et al., 2017) three layers architecture which contains cloud, Fog, and the physical nodes, have been proposed to handle smart home challenges. The Fog layer abstracts the communication between the cloud and the physical devices which communicate using ICN paradigm. The Fog layer also conceals devices heterogeneity, and supports real-time communications. Another separation vision has been proposed by authors of (Yue, et al., 2014)**,** where users are grouped according to their interests to form a logical groups or communities known as the logical space. These logical communities are physically connected by using Forwarding and Caching Nodes (FCN) and Community rendezvous Points (CRP), which form the physical space. The whole idea of the logical space is to guarantee resilient dissemination of data. Cloudlets of communities which represent a fog plane manage all community's issues such as creating, and deleting communities.

2.2 Privacy in ICN Architectures

Few numbers of researches in the literature have studied security and privacy in ICN. The following is a brief discussion of privacy issues as mentioned in different proposed architectures.

CONVERGENCE has identifies a set of security and privacy objectives so that any future implementations should include these objectives. Security services are expected to be implemented by Technology Engines (TEs) in the CoMid supported by functions implemented in the CoSec. CONVERGENCE defined only two objectives regarding privacy which are protecting publisher's and subscriber's identity (Salsano, et al., 2011), (Melazzi,

2011).

C-DAX defines four security objectives: authentication of users, topics access control, end-end integrity, and confidentiality. SecServ authenticate publishers and subscribers as well as other system nodes, and manages keys distribution at joining process. C-DAX allows using traditional security mechanisms and does not restrict certain cryptography techniques. In (Sicari, et al., 2017) security was considered on four levels without specifying privacy: data and service delivery, service discovery, naming, and subscribers/publishers registration. Mainly authentication and confidentiality are considered.

Authors of (Yue, et al., 2014) have built their security services based on trust model, they assumed that CRPs are trusted nodes and FCNs are semi-trusted nodes. They suggested protecting privacy by using attribute-based encryption techniques but they still need more investigation for their complexity. The authors also proposed a secured signature to authenticate and verify the integrity of contents. Similarly, the architecture in (Chen, et al., 2016) protects privacy using attribute-based encryption, users are authenticated and contents integrity are verified using signatures, unauthorized users can be rejected based on chain of trust.

As can be observed so far, the contemporary architectures usually are more sophisticated than being flat topological architectures, due to the new applications domains requirements and their extended challenges that imposed themselves. Among these challenges and the reasons why it becomes mandatory to consider them are: (1) mobility, because of the increasing number of mobile connected nodes, (2) scalability, because of connectivity expansion, (3) modifiability, since that technologies are evolving by time and new components are rapidly taking the place of old ones, (4) heterogeneity in SW and HW, because of the developing companies that entered the technology industry with different standardization and specifications, as well as, integrity with legacy technologies, (5) All previous points should not be considered without taking security as a key point in any architecture design, due to the increasing number of cyber-attacks, security threats and their variety in technology and motivations. In the next section a detailed description is presented about the proposed privacy-aware ICN-IoT architecture with scenario-based discussions of the different communications. These scenarios are also formalized using situation calculus and sequence diagrams to analyze the overall framework of communication privacy services.

3. PPUSTMAN: *P*rivacy-aware *PU*blish/*S*ubscribe Io*T* *M*VC *A*rchitecture using Information Centric *N*etworking

This section explains in details the proposed ICN-IoT architecture PPUSTMAN which consists of smart data plane with name resolution, data forwarding, and controlling functionalities to provide network administration flexibility. It uses Publish/Subscribe (Pub/Sub) ICN communication approach to support mobility, and network performance enhancement. This approach of communication decouples requesters from providers in location and time, which provides efficient control of data transfer, and can provide privacy for both publishers and subscribers in an easier and more flexible way (Jin, et al., 2017). Moreover, PPUSTMAN also enhances ICN Pub/Sub approach by implementing the Pub/Sub network components interactions through Model-View-Controller (MVC), which follow the MVC design pattern (Reenskaug, 1979). MCV increases the number of communication layers to provide more abstraction, concurrent communications, and maintainability (GuangChun, et al., 2003). An overall view of the PPUSTMAN architecture is shown in Figure 1 and will be explained in details in the next subsections.

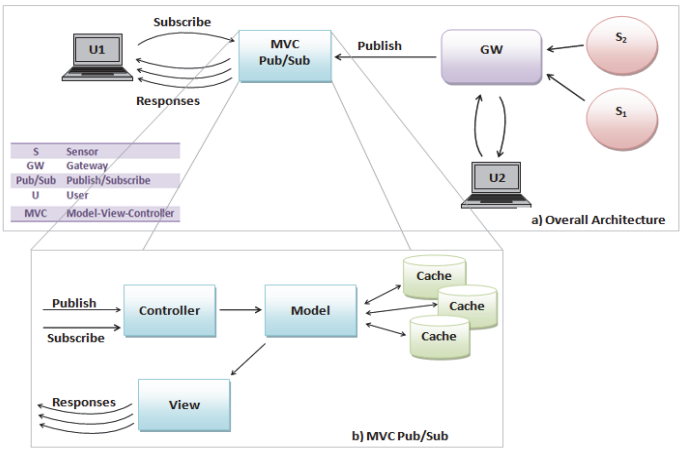

Figure 1. MVC Pub/Sub architecture, a) Overall Architecture, b) Details of Pub/Sub component

3.1 Publish/Subscribe Model-View-Controller Architecture

Figure 1 (a) shows overview of IoT communication scenario where a gateway collects information from different sensor nodes, and publishes this information through the MVC Pub/Sub component. Meanwhile, subscribers send subscription messages to controllers to request information, and then the Pub/Sub component pushes the information once it is available. The Pub/Sub component as shown in Figure 1(b) consists of three main entities: Model, View, and Controller. Caches are devices with powerful caching capabilities that support the Pub/Sub functionality, their existence in the architecture are preferable but not mandatory since it is a deployment decision that depends on the available ICN infrastructure. The architecture also contains different types of nodes with or without caching capabilities, such as publishers and subscribers. Table 1 summarizes the functionality of each entity.

PPUSTMAN consists of clusters that represent different domains. Each domain has a Controller, View and Model components. The Controller is responsible for user's authentication, checking privacy requirements, name resolving within its domain, cooperating with other domains controllers using name-based routing, and forwarding interests to the model component. The Model component is responsible for caches orchestration including caching policies, preparing contents to be sent to the subscribers, encrypting information if privacy policies require encryption, and notifying the View component to deliver information to users. The View component, applies routing aggregation if required, and determines routing paths to users according to privacy and performance policies.

Table 1. PPUSTMAN Entities

ENTITIES	*FUNCTIONALITY*
Controller	The controller receives sub/pub messages, authenticates users, performs privacy aware actions, redirects published data to the Model, maintains registry table, and performs name-based routing and name-resolution. It also resolves the heterogeneity between user's networks.
Model	Each domain has a Model that acts as caches orchestrator, performs data-preprocessing. It also encrypts private data, performs adapted privacy-aware caching policies, and interacts with the View component.
View	The View component resolves the heterogeneity between user's networks, performs adaptive routing protocols taking into consideration privacy protection and network performance.
Publisher	Publisher is a node that publishes data, provides services, and issues actuating commands.
Subscribers	Subscriber is a node that subscribes to retrieve certain contents, services, or actuating commands. Subscribers also can pull information from the network.
Caches	Caches are main storage devices that store data off-the path, using different caching policies defined by the Model component. Moreover, contents can be cached along the routing path using in-path caching policies which are performed by nodes with caching capability.

PPUSTMAN inherits the advantages of MVC pattern. For instance, instead of having direct communication with caches or gateways, users have to interact with caches through controllers and models, consequently, allowing a level of abstraction between each layer. Another inherited advantage is that this layering methodology allows more concurrent communication flow. For instance, while controlling commands or requests are directed from controllers to models, responses by caches can be routed from models to clients through view components rather than going back through controllers as what happen in the N-Tier architectures. This allows controllers to focus on monitoring and name resolution functionalities, therefore, relieving controllers from routing and caching issues. Decoupling between users and models leaves interoperability with users as the responsibility of controllers and views. Accordingly, reduces the overhead on models, and supports handling heterogeneity between IoT devices. The existence of the model layer abstracts the interaction between controllers and caches. In this way caches can be based on various platforms that are implemented by different vendors. In the same way, the abstraction provided by the architecture in the other layers allows flexible replacement of devices from new vendors by administrators.

PPUSTMAN is a privacy-aware architecture which takes privacy issues into consideration. For instance, one can imagine PPUSTMAN domain as a sphere where caches are hidden in the inner core, as shown in Figure 2. Hiding the most important components and information within the inner components allows the architecture designers to apply different security mechanisms to secure these parts, and preventing direct interactions with

them. Similar to any ICN architecture, PPUSTMAN stores the responses of requests using in-path caches, which are located on the routing path between View and the subscriber, unless this violates privacy policies. Additionally, communications can be either stateful or stateless, depending on the privacy policies adopted by architecture administrators or users. Location and identity privacy can be supported by decoupling between subscribers and publishers, where subscribers are interested in information despite of their providers, and publishers publish data without being aware of data subscribers or their locations.

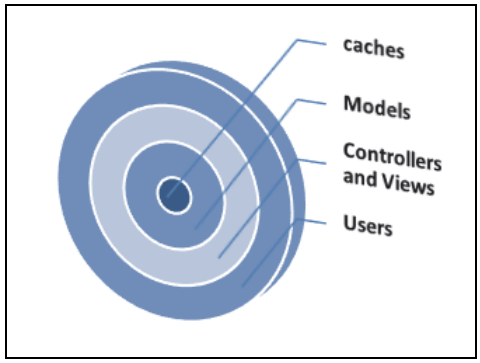

Figure 2. PPUSTMAN domain layers

There are two types of caches in PPUSTMAN: the first type is main caches managed by the model entities as illustrated in Figure 2, and in-network caches within nodes which have the well to share their caches such as user's caches. Caches are handled by Pub/Sub components, and specifically the Models. Meanwhile, in-network caches perform data caching along the routing path (in-path). PPUSTMAN keeps the main caches in the inner level of the architecture to protect their data, allowing only models to communicate with them. Main caches can be categorized according to the type of data they cache. One of the caches stores the most private information, while the others store other data according to data trust evaluation.

Figure 3 shows inter-domain interactions in PPUSTMAN where each domain provides information to its requester indirectly through the Pub/Sub components. Implementing network clustering has proved its advantage in supporting more scalability compare to centralized systems. It also preserves constrained device's resources such as energy, which in turn increases the life time of wireless nodes (Mamun, 2012) (Carlos-Mancilla, et al., 2016). Inter-domain communication in PPUSTMAN is managed by controllers and views components. Subscriptions that cannot be served by local controller will be routed by the controller to other cluster's controllers using NBR mechanism. Once a controller that can serve the subscription is found, the contents are forwarded to its subscribers through views components.

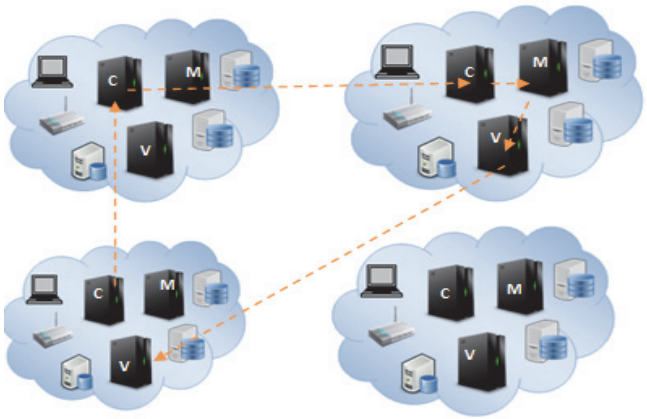

Figure 3. Inter-domain communications

3.2 Communication Scenarios

This section explains in details publishing and subscription communication scenarios such as publishing actuation commands to actuators. Different messages and operations used are summarized in TABLE 2 and

TABLE 3 respectively:

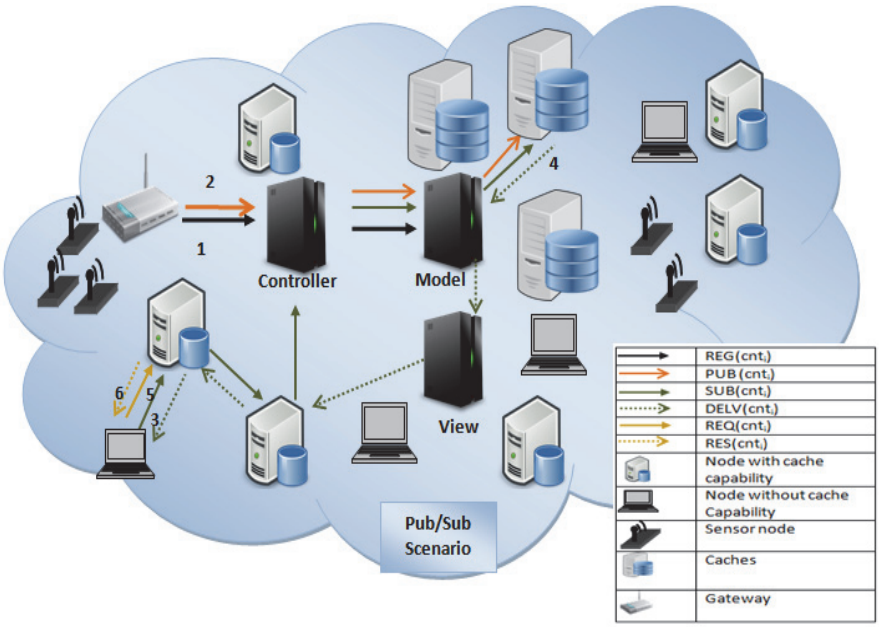

Figure 4. Content Pub/Sub scenarios

1. Registration: In this scenario, a gateway collects information from sensor nodes, and then performs certain required preprocessing such as aggregation, and filtering. It then sends a Register message (REG) to the controller, as shown in step (1) in Figure 4, in order to register the name of the topic or contents. The REG message contains content or topic unique name, required privacy service, and other criteria such as access control rules defined by the publishers. The controller checks privacy service defined in the message, updates its registry table, and then forwards the REG message into the model. The model checks meta data and privacy service required, prepares its registry table which includes cache-name mappings, to determine in which cache the information will be stored. Meanwhile, the controller checks whether there are pending subscription requests from other clusters, if yes, then it informs the controller which requests the data, that the information can be supplied by the view component as soon as it is available.
2. Publishing: Publishing data scenario as shown in Figure 4, step (2). In this scenario, the gateway starts sending data within Publish messages (PUB) to the controller event based or periodic based. The controller in turn forwards the PUB message to the model to cache the data according to the caching policy determined.
3. Subscription: Subscriptions start as shown by step (3) in Figure 4, when users send a subscription message (SUB) to show their interest in certain content, or all contents published under certain topic. SUB messages include content's unique name, privacy service, and any other criteria defined by the subscriber such as some filtering or data aggregation. The SUB messages are sent to the controller which verifies its registry table to check whether the requested topic is within its scope, if yes, then it forwards the SUB message to the model. The model retrieves the contents from the caches, as shown in step (4). Then it performs the required security mechanisms such as data encryption, and notifies the view to start the delivery process. The view determines the routing path to forward the contents to its subscribers, performs any required routing aggregation, takes network conditions and privacy into consideration, and then it pushes the contents to users using PUSH messages. This forwarded information can be cached along the route, in-path caching, to be used in future requests of the same information. If the controller could not find the name defined in the SUB message within its scope, it starts contents discovery process using inter-domain NBR with other neighboring controllers. Once the contents are found, the providing domain informs the current controller, and starts delivery process through the providing domain view and the requester domain view components as it shown in Figure 3. Step (5) and step (6) in Figure 4 show that the network retrieves the response for any information Request message (REQ) from the nearest cached copy, assuming that the previous responses were cached before, otherwise the response will be fetch from the source.

4. Actuation: It is important to consider actuation when designing any IoT architecture. Authentication is a core process in actuation because actuators control other devices, and it is important to be sure, that only authorized parties control these actuators (Lindgren, et al., 2014). For instance, an actuator might be part of a health system that controls patient's drug dose. Such complicated environments where real-time, highly secured communications, have to be considered in the earlier phases of the architecture design. In PPUSTMAN architecture, actuators are considered as subscriber nodes, while users controlling these actuators, are considered as publisher nodes. Figure 5 shows the actuation steps, where users send REG messages, as shown in step (1) of the figure, to inform controllers about their interest to issue new commands. Publishers in this scenario should be authenticated before completing registration process, step (2) of the figure. Once the authentication is successful, publishers can start publishing their commands as shown in step (3) of Figure 5. An actuator or its gateway sends SUB messages, illustrated by step (4) of Figure 5, to inform the controller about their interest in receiving actuation commands. Subscribers are also authenticated as illustrated in step (5), before receiving actuation commands. Then finally step (6) where commands are pushed to authenticated subscribers.

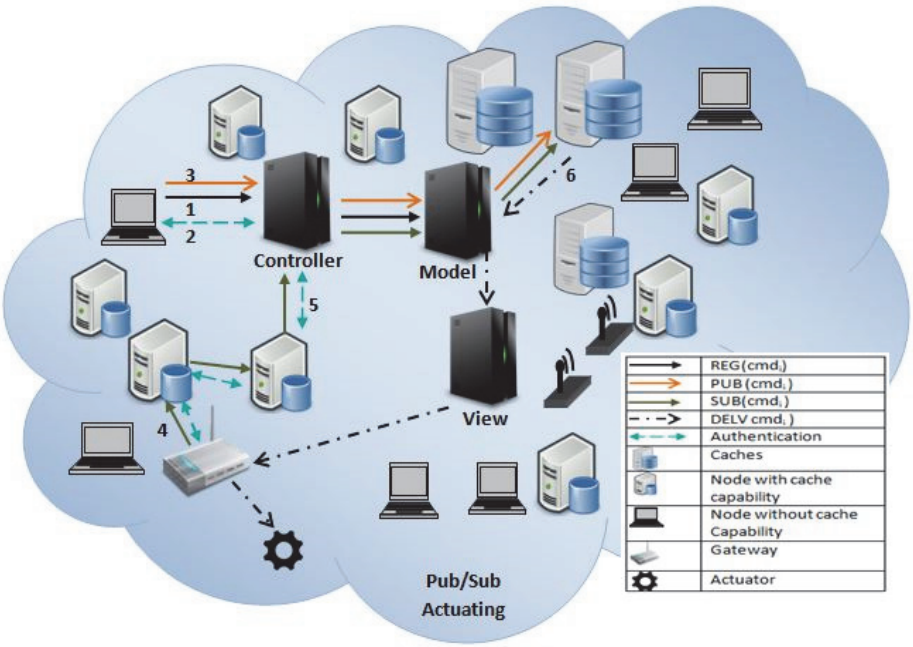

Figure 5. Actuation Scenario

Table 2. Pub/Sub Mvc Architecture Messages

MESSAGES	*MSG FLOW*	*Functionality*
PUB	Publisher - Controller	Sends available data uniquely identified under specific topic.
REG	Publisher - Controller	Registers a new topic to publish data under this topic, or register a name for certain content.
SUB	Subscriber - Controller	Declares interest of certain content or all contents under specific topic.
REQ	User – Network	Requests certain data by users.
RES	Node with cached content – User	Responses to REQ messages.
REVOK	Publisher - Controller	Deletes certain content or topic.
PUBEND	Publisher - Controller	Unregisters certain topic, notifying the end of publishing.
REVOKS	Subscriber - Controller	Unsubscribe certain topic or content.

UPD	Publisher - Controller	Modifies a content name, or certain content's values.
NOTIF	Model - View	Notifies view with caches changes and to prepare delivery.
FETCH	Model - Caches	Requests information from caches.
FIND	Controllers - Controller	Inter-domain NBR to find certain content or topic.
PUSH	View – Subscriber Model - Caches	Sends information from View subscribers or from Model to Caches.
ACCEPT/REJECT	View-Publisher View-Subscriber	Accepts or rejects a subscription or publishing.

Table 3. Architecture Main Operations

OPERATION	*Functionality*
AUTH	Authentication operation.
MODF	Modification of caches and registry tables operation.
ENCP	Encryption operation.
CHECK	Checking certain constraints such as privacy, authentication requirements of a message, or caching policies for some registry.
PREP	Determination of routing path.

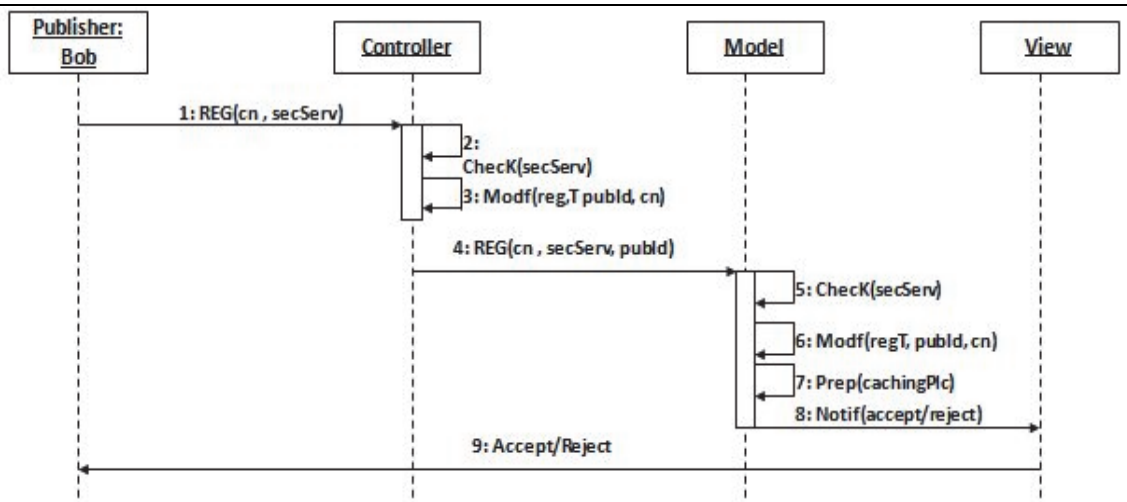

Figure 6. Registering contents scenario

Figure 6 to Figure 9 are sequence diagrams that show the details of data registration, data publishing, data subscription, actuation issuing commands, and actuator subscribing for commands scenarios. The first sequence diagram is detailed in Figure 6, where publisher Bob would like to register certain topic at the controller of its cluster. Bob starts by sending $REG(cn, secServ)$ message with content name (cn) and security service ($secServ$) required by Bob for this content including privacy requirements, access control roles, or authentication issues. Once the controller receives the *REG* message it checks the security service required using $Check(secServ)$. Then by assuming in this scenario that Bob needs no specific security service, the controller modifies the registry table ($regT$) by adding new record with publisher Identification ($pubId$) and the registered content name (cn) as shown by $modf(regT, pubId, cn)$. This information is forwarded to the model which in turn checks $secServ$, prepares the required caching policies and chooses a cache to save the coming content, and then modifies its registries. The model notifies the view to accept the registration, or to reject it if any violation is found such as "caches no empty space" or "un-trusted behavior detected".

Figure 7 shows the details of registering actuation commands where publisher Bob would like to register commands for AC actuator device at his office. Bob starts by sending *REG* message to the controller with $REG(cn, secServ)$. Since only authorized users are allowed to issue actuation commands, the controller

authenticates user Bob first. After successful authentication the controller activates the $modf(regT, pubId, cn)$ to modify its registry table then it forwards *REG* message to the model. As in the previous scenario the model checks information in the *REG* message to choose caching policies. Then an accep/reject message is sent through the view.

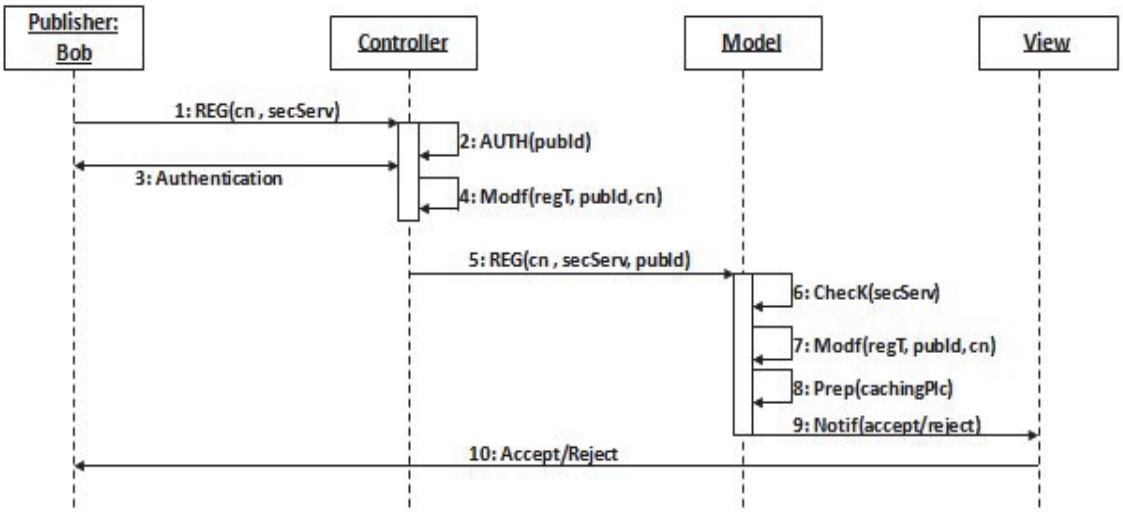

Figure 7. Registering actuation commands

Authentication required for publishers is performed once at registration time, and then publishing process starts after authentication acceptance to reduce traffic load. Unless, highly secured data transfer is required, in this case, an authentication each time data published is required. Publishers would require secure publishing for their data, this choice is shown in Figure 8 with two alternative sequence scenarios, one with secured data caching and the other without securing data caching. Publishers send $PUB(NDO_i)$ messages to the controller on each event occurrence or on periodic based. The controller checks whether the name of these NDOs are registered or not. Then it forwards the message to the model to encrypt information and store it in one of the chosen caches. The alternative choice in the figure shows the steps without encryption of NDOs.

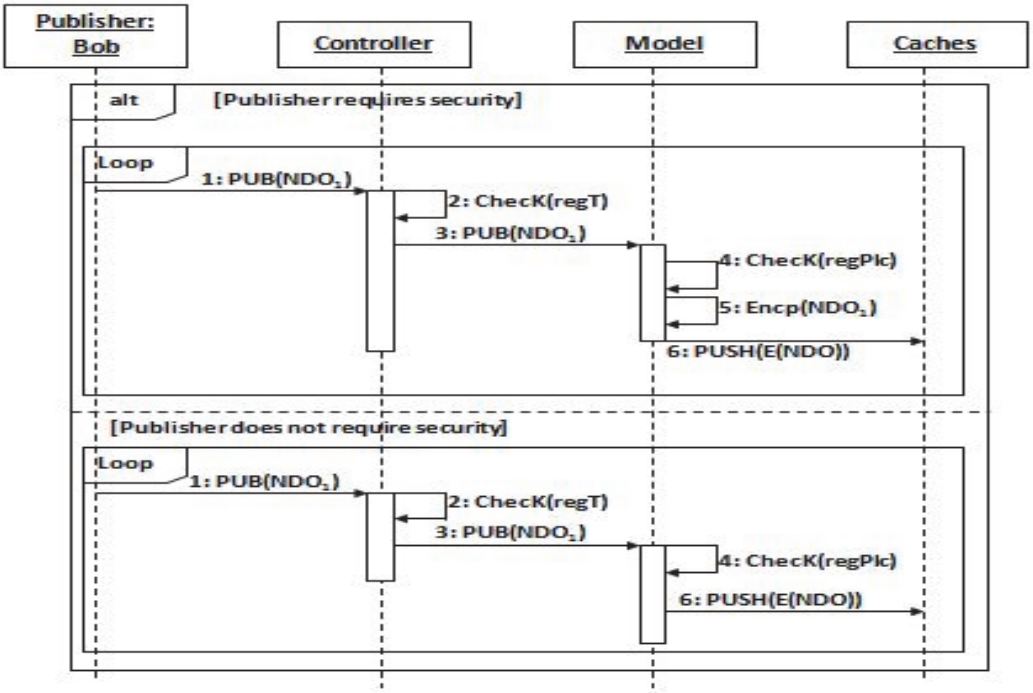

Figure 8. Publishing contents or actuation commands scenario

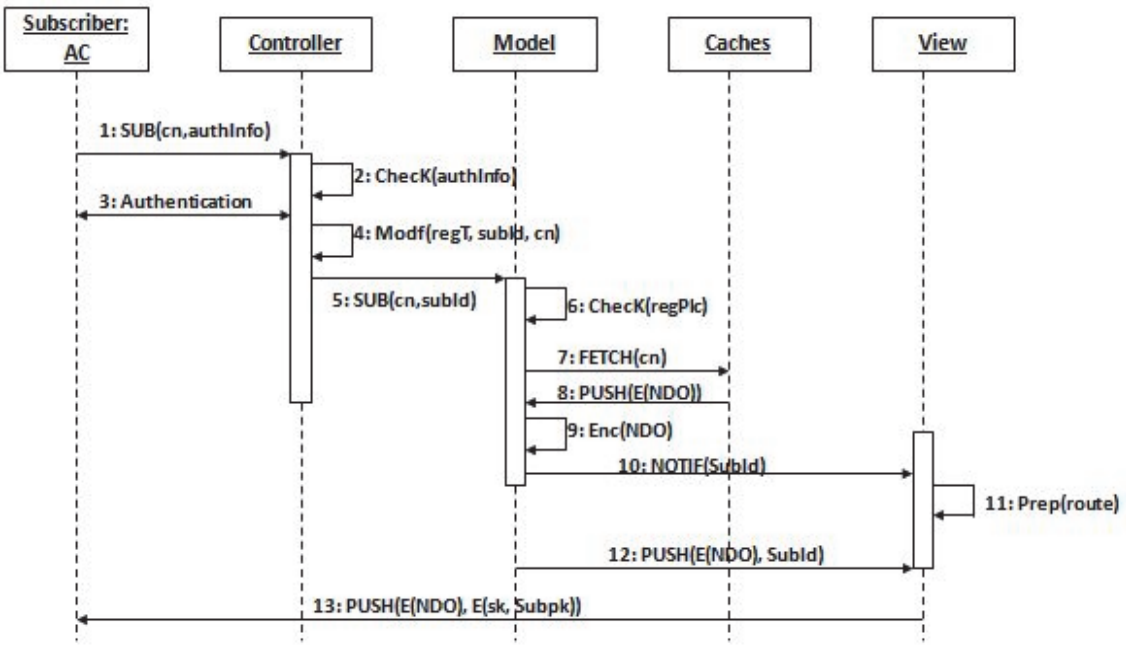

Figure 9. Subscribing for contents or actuation commands scenario

Subscribing for data or receiving commands steps are shown in Figure 9. For the reason that, receiving commands should be claimed by authorized subscribers only, controllers authenticate subscribers before sending the commands they request. Then controllers forward the *SUB* messages to the model to fetch the information from the caches, notify the view to determine delivery route and then to start sending packets to subscribers through the view component which choose privacy-aware routing paths.

3.3 PPUSTMAN Goals

PPUSTMAN architecture takes into consideration the following goals and challenges:

- Orchestration: (1) Caches orchestration where aches placement and selection policies are managed and adaptively configured by the Models. (2) Caches choreography where distributed caches inside nodes cooperate and work in the way designed by the underlying ICN paradigm. (3) Cluster orchestration is implemented by controllers since a controller is considered as the cluster head which manages and monitors cluster performance and takes some highly intelligent decisions.
- Performance and its related issues: (1) Heterogeneity issue is handled by the abstraction in different levels. For instance, requests are passed to models indirectly through controllers. Meanwhile, responses are passed to clients through views. Controllers and views increase interoperability by abstracting the communications between Pub/Sub and models on one hand, and by hiding the details of caching policies behind a Model on the other hand. Furthermore, views adapt routing protocols according to client's network paradigms. Moreover, controllers handle the heterogeneity between clusters which could be implemented using different networking paradigms. (2) Scalability issue is handled by clustering the Pub/Sub architecture which is known for its scalability weakness (Yoo, et al., 2009). Consequently, Pub/Sub architecture performance might be worse in IoT environments. To handle scalability issues PPUSTMAN suggests splitting the network into distributed clusters each with pub/sub functionality, (3) Delay and network traffic load are enhanced by the concurrent communications by applying the MVC communication pattern. While controllers handle new coming messages, models interact with views to send responses. ICN by nature allows routing aggregation and caching performance, (4) Limited resources awareness is required in the context of IoT since it is known for its constrained devices, and accordingly, light-weight protocols and security techniques are needed to preserve these resources. Pub/Sub pattern and in-network caching of ICN can preserve energy consumption. Heavy encryption and authentication mechanisms can be carried out in controllers, views and models which are considered as powerful nodes. (5) Mobility issue basically is supported in ICN by using NRS. Problems of mobile publisher who move into other clusters is handled by the cooperation between clusters controllers in exchanging the contents published by that mobile publisher. Mobile subscribers notify the Controller of the new cluster, asking it to negotiate with the old controller

about its subscription information. And whether the requested information is available within the new cluster or it is required to be forwarded through both cluster's Views or not.

- Security and its related issues: (1) Trust is managed in different aspects, for instance, subscribers can be trusted according to their requests behavior, such as intensive subscription which might be considered as a DoS attack. Contents could be evaluated by users to determine their quality and their trust level. Content's trust levels affect their publisher's reputation, and in-network caching reputation, which might affect a whole cluster reputation. Moreover, main caches are categorized to contents high-trusted caches, and contents moderate-trusted caches. (2) Authentication is handled by Controllers which authenticate authorized publishers and subscribers, specifically, those who are involved in issuing commands for actuators. (3) Publishers define their own access control policies for their published contents. (4) Privacy related issues are handled by the architecture in many aspects which are summarized in Table 4. Privacy policies should be adaptive according to privacy-requirements. For instance, subscribers issue stateless subscription messages to prevent controllers or views from keeping records about them. Publishers inform controllers whether to treat them as private publishers, which in turn demands controllers to secure or hide their identity. Moreover, publishers might inform controllers whether their contents are private contents or not, and whether these contents need to be encrypted before caching or not. Location privacy can be preserved as a countermeasure against different attacks. One of these known attacks is the timing attack, where attackers calculate responses Round Trip Times (rtt) to predict how far data sources are, or whether they are mobile nodes or not.

Table 4. Implementation of Privacy Aspects

Privacy Aspects	*Implementation*
Private Routing	Implemented by Views.
Private data	Encryption in Model, using secured caches only to store data, decoupling publishers and subscribers from model and caches.
Name Privacy	ICN-Naming scheme.
Users Identity	Decoupling publishers from subscribers, and decoupling subscribers and publishers from the model. And using anonymity techniques.
Location Privacy	Using naming schemes rather than IP addresses, Pub/Sub pattern decouples publishers and subscribers in location. As well as, using secure routing.

4. Analysis and Discussions

4.1 Architecture Analysis

To compare between N-Tier and MVC architectures, two sequence diagrams for overall registration, publishing, and subscription scenarios are shown in Figure 10 and Figure 11. N-Tier architectures are the standard methodology of passing messages. In web development N-Tier usually consists of User Interface (UI) which is deployed at user's devices, Business Logic (BL) component which is usually deployed at servers, and Data Access Layer (DAL) component which is responsible for abstracting DB and is either deployed on special server or with the DB server. For fair comparisons we assumed that each tier in the N-Tier architecture is deployed on different server similar to our MVC architecture. Table 5 shows a comparison between number of communications and number of internal processing performed in both architectures based on a full publish/subscribe scenario in N-Tier and MVC as illustrated in Figure 10 and Figure 11 respectively.

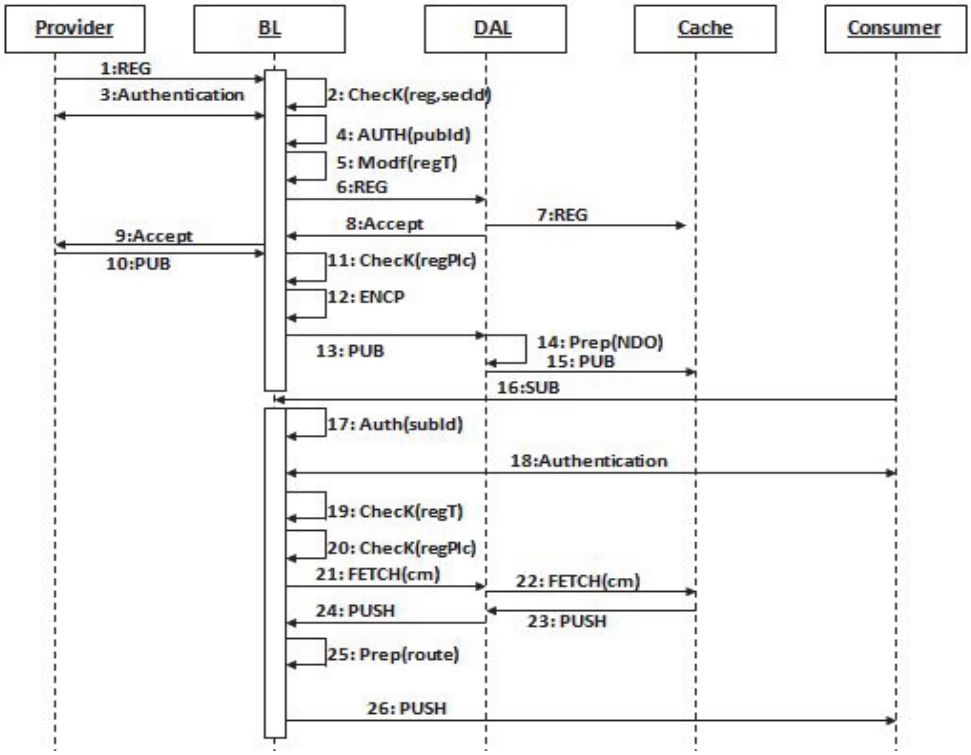

Figure 10. N-Tier Architecture

Table 5. Comparisons between N-Tier and MVC architectures

Component	*Architecture*	*Communications*		*Processing*	
Overall	N-Tier	16		10	
	MVC	16		11	
Controller	N-Tier	12		9	
	MVC	8		6	
Parallel on Controller	N-Tier	Reg. & Pub	Sub	Reg. & Pub	Sub
		7	5	5	4
	MVC	Reg. & Pub	Sub	Reg. & Pub	Sub
		5	3	4	2

As observed from the comparisons in Table 4, the communication cost, represented by the number of exchanging messages between parties, for both N-Tier and MVC is the same. On the other hand, processing cost, represented by the number of processes activated by all parties, is slightly better in N-Tier by difference of one process. By analyzing the cost of communications on controller of the MVC or BL of N-Tier, it can be observed that controller's communication and processing costs are both enhanced in the MVC by 40%, comparing to BL of the N-Tier. The reason of this enhancement is that some of the BL's functionalities are performed by the model or the view components in MVC. Distributing functionalities on the MVC components reduces the overhead on each component, and adds the space to perform parallel tasks. The last part of the table shows the number of parallel processing on controllers in publishing and subscription scenarios for both architectures. And it shows that MVC outperforms N-Tier by two points in most cases.

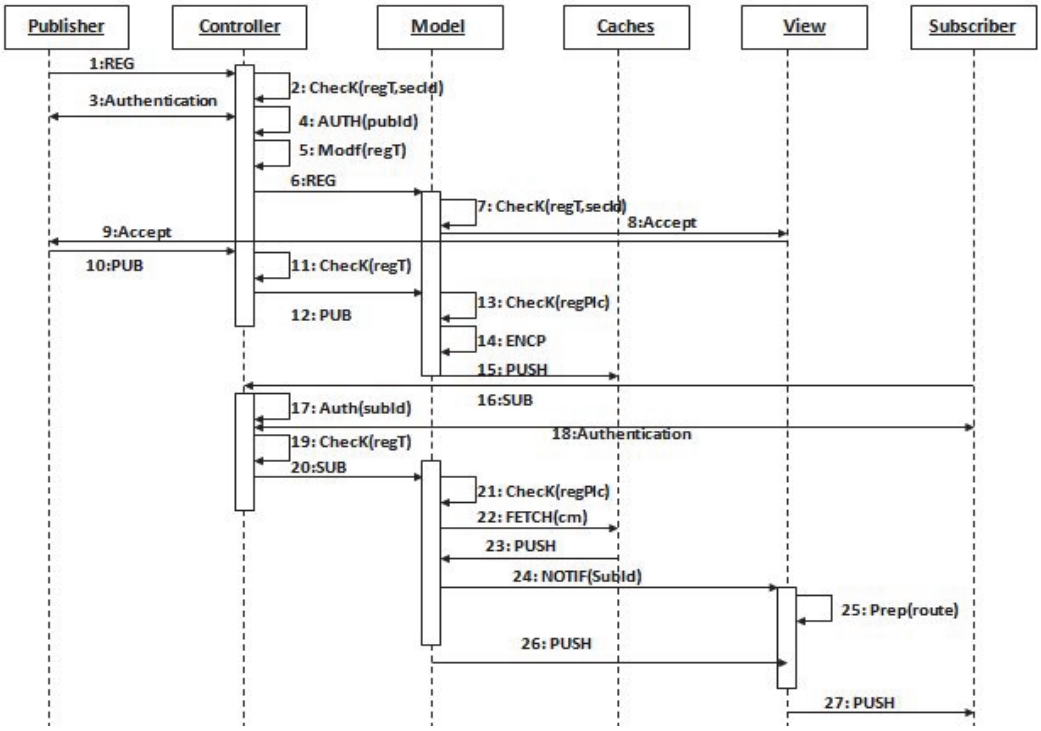

Figure 11. MVC architecture

4.2 Communication Analysis

The following is an analysis of the communication cost of different communication scenarios:

1. Event-data retrieving using pull-based communications: Figure 12 shows a simple direct communication between sensor nodes and users, where users use pull-based mode to retrieve information form sensors. Each time a user need to retrieve a reading from a sensor he issues a request message *REQ* and the sensor responds with a response message *RES*. Equation 1 calculates the cost of exchanging *M* messages between set of *U* users and *S* sensors. Table 6 lists symbolic meaning of Equation 1.

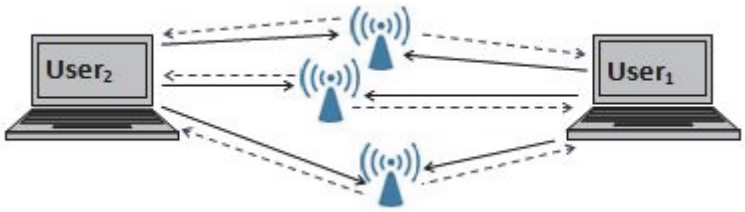

Figure 12. Clients-sensors pull-based communications

$$Cost = \sum_{i=1}^{U} \sum_{j=1}^{S} \sum_{k=1}^{M} \left(CRQ_{ijk} + CRS_{ijk} \right) \quad (1)$$

Table 6. Equation 1 symbols key

Symbol	*Description*
Cost	Total communication's cost
U	Number of clients.
S	Number of sensors.
M	Number of messages.

CRQ_{itk}	Cost of the k[th] request message between $user_i$ and $sensor_j$.
CRS_{ijk}	Cost of the k[th] response message between $user_i$ and $sensor_j$

This communication pattern increases traffic load on the network for periodic data retrieval, resending the same information to multiple users is a burden on constrained devices such as sensors. Furthermore, communication duplicates which are caused by resending responses to the same user or other users, increase traffic load of the network. This communication scenario does not support sensors or client's mobility. As well as privacy, especially location privacy is not easily protected in this scenario.

2. The previous communication scenario in step (1) can be expanded by using gateways or application servers. Gateways or servers gather sensors information and apply data preprocessing before responding to requests. Users send requests to the gateway or to the server connected, with their desire to receive future data generated by sensors. In other words, users subscribe for certain data, and servers push the data as soon as it is available. A Pub/Sub communication pattern shown in Figure 13 reduces the traffic load on the network compared with that of Figure 12. Equation 2 calculates the communication cost, UCost, from the gateway or server side to users, where each user sends a subscription message that includes the names of the sensors which have the required information. Equation 3 calculates the cost, SCost, from each sensor side to server, as can be observed the traffic load is reduced significantly from both sides. Table 7 lists symbolic meaning of Equations 2-3.

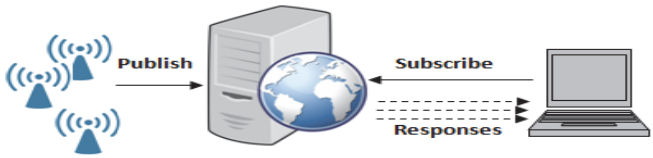

Figure 13. Pub/Sub communications

$$UCost = \sum_{i=1}^{U} CRQ_i + \sum_{i=1}^{U}\sum_{j=1}^{M}(CRS_{ij}) \quad (2)$$

$$SCost = \sum_{i=1}^{S}\sum_{j=1}^{M}(CRS_{ij}) \quad (3)$$

Table 7. Equation 2 and 3 symbols key

Symbol	*Description*
UCos	Total communication's cost between users and server
SCos	Total communication's cost between sensors and server
U	Number of users
S	Number of sensors
M	Number of messages from sensors to server
CRQ_i	Cost of request message between $user_i$ and the server
CRS_{ij}	Cost of response message between $sensor_i$ and the server and from server to $user_i$

In this scenario, if users try to retrieve information that is already sent to them or to any node in the neighborhood, the request message is responded to by the original server. Retrieving information from the original server for each request of the same information overloads the network traffic, which is enhanced by

using in-network caches as explained in case (3).

3. Using ICN in-network caches as shown in Figure 14 reduces network traffic overhead, where *userA* requests data for the first time and the response comes from the information server directly, and in-path caches store information for subsequent requests. When *userB* requests the same data, the response can be served by the nearest cache rather than going to the original server. Consequently, this reduces traffic load over gateways or information servers. Network load enhancement depends on the probability of cache's hits, and the replacement policies applied in each cache.

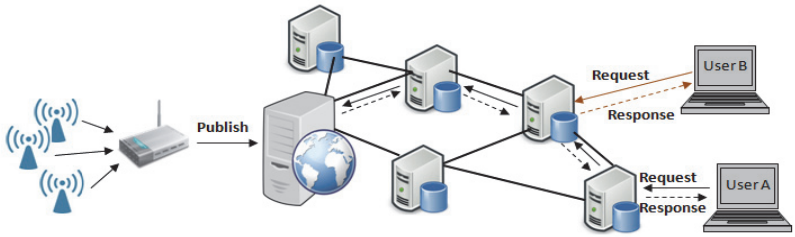

Figure 14. In-Network caching

The following mathematical model represents an estimation of data requests or contents hit probabilities in caches, where hit probability increases proportional with request frequency.

$$h(c_i) = \begin{cases} 1 & ,N \leq cn \\ RF(c_i) & ,N > cn \end{cases} \quad (4)$$

$$RF(c_i) = \frac{\#\ of\ c_i\ requests}{Max(requests)} \quad (5)$$

Equation 4 calculates the hit probability $h(c_i)$ of contents availability in caches, where c_i is the i^{th} content stored in the cache, N is the total number of generated data in a period of time. For simplification, assuming that contents are of equal sizes, then the number of contents that can be fit in the cache (cn) can be calculated by dividing cache size s by content size cs. $RF(c_i)$ is the request frequency for content c_i which is calculated by Equation 5.

4. Hits of IoT historic periodic contents are more complicated to model, since large amount of data is generated each specific interval of time depending on the application's domain. For instance, news reporting organization publishes news each day and at each event. Another example, is sensors that monitor patient pulse, they publish data nearly each minute. In both examples, many users might be interested in some historic information which was requested by the same users or other users. Caching such information, enable the network to response to later requests from the nearest cached copy. Equation 6 calculates requested content's hit $h(c_i)$ for periodic data, where uniformly random distribution for contents insertion mechanism is chosen. The first expression in the equation represents a non-full cache, where cache capacity is enough to store all generated information so far. The second expression represents a full cache where the cache capacity cannot store all generated contents, and replacement mechanism should be chosen. In this case, contents have the same probability for being replaced.

$$h(c_i) = \begin{cases} 1 & ,N < cn \\ \frac{1}{N * CN} & ,N \geq cn \end{cases} \quad (6)$$

Meanwhile, Equation 7 calculates $h(c_i)$ where the replacement policy depends on three factors: freshness of data $(\frac{i}{N})$, frequency of requests $FR(c_i)$, and cache size $(\frac{cn}{N})$:

$$h(c_i) = \begin{cases} 1 & ,N < cn \\ FR(c_i) * \frac{i}{N} * \frac{cn}{N} \times 100\% & ,N \geq cn \end{cases} \quad (7)$$

Figure 15 and Figure 16 plot the hits probability calculated by Equation 7 for caching generated sensor data in 1 hour period, where the sensor generates information each one minute. For simplicity, we assumed that each information size is 1 KB, and c_i is the content generated at minute i. As can be observed from Figure 15, that hits probability increases proportional with cache size, the more the cache size the more the probability to find certain content. Any caching policy in IoT should take into consideration freshness and popularity of data, where popularity can be represented by frequency of requests. Figure 16 shows the relation between caching hits and request frequency. As expected the more frequent any content is requested the more the probability to find it in the cache. Both Figures show the relation between freshness of data contents and their hits, this is important for any caching policy to give recent generated data priority for being cached larger amount of time, because it is possible to be requested by users.

The relation between network traffic enhancement and using caches for periodic data requests is represented in Equation 8, where the enhancement of network *NE* is increased by a percentage of success hits average.

$$NE = \frac{\sum h(c_i)}{N} \quad (8)$$

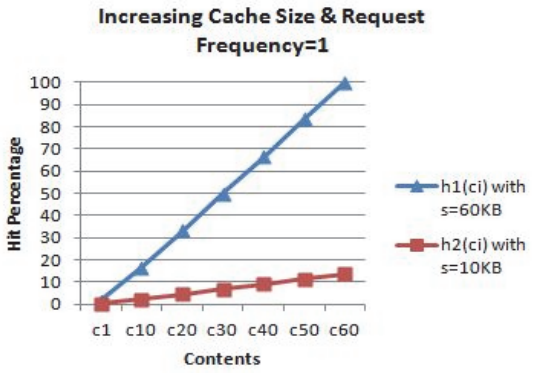

Figure 15. Relation of cache hits and cache size

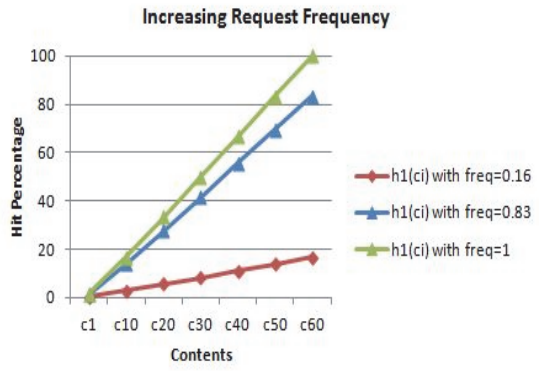

Figure 16. Relation of cache hits and requests frequency

Caching issues such as placement selection and caching insertion policies are the major impact on network performance. Different caching policies exist in literature to enhance network performance and QoS of systems. In (Jin, et al., 2017) the authors have surveyed caching policies such as caching everything everywhere (CEE), caching with probability (Prob(p)), breadcrumbs, hierarchical cooperative caching (HCC), and many others. Some of these policies depend on caches cooperation among each other to provide information. For instance, Partial Cache policy divides large information such as videos into large-grain of chunks and cached it with groups of neighbor caches. On the other hand, some caching policies do not require caches cooperation such as Prob(p) where caching a content is a decision made by each cache depending on its popularity (Jin, et al., 2017).

4.3 Formalization and Reasoning of Communication Scenarios Using Situation Calculus

Situation calculus (Levesque, et al., 1998) is first developed by McCarthy (1963) and re-introduced by Reiter (1991). It represents world changes using First Order Logic (FOL) formulae. The main components of situation calculus are: (1) actions which occurred in a situation and change it into other situation, such as $putOn(x, table)$ which represents the action of putting object x on the table (2) fluents which are domain-dependent predicates, there are two types of fluents: relational fluents which represents the truth of a state such as clear(table) which means that the table is clear, and functional fluents which returns a situation-dependent value, such as color(wall) which returns the wall color, (3) situations (S) which are represented as world states by McCarthy, while Reiter described a situation as a finite sequence of actions or history such as [open(),enter(),walk()]. There are another domain independent predicates which are the *Poss* and

do, ***Poss(a,s)*** represents the possibility of any action *a* occurrence in situation *s*, for instance $Poss(hold(x), s) \rightarrow next(x)$, and ***do(a,s)*** which represents the situation which occurred after performing action *a* within situation *s,* for example do(close(x),s) means that the current situation occurred after performing a close action in previous situation s.

Upon action occurrence, some states might change which can be expressed by affect axioms. For example $isOpen(door, do(open(door), s0))$ implies that the door is open in current situation, s1, because of $open(door)$ action that occurred in previous situation s_0. Representing changes of fluent's states, adds causality and temporal flavor to situation calculus, which enabled using situation calculus for problems detection applications. Reasoning can be implemented in situation calculus by inferring the set of axioms starting from the initial situation S_0 reaching the goal or target situation, which is known as forward reasoning. Forward reasoning can be used in planning situations, this is expressed in situation calculus extensions languages such as GoLog, which defines more complex components such as if, proc, and while to be used in planning.

Situation calculus formalization is adopted in this research to describe the communication pattern in subscription, publishing scenarios, and the alternate actions occurred within these scenarios. Moreover it describes the system events and communication in a temporal framework, i.e. receiving msg_{ik}action can never occur before sending msg_{ik}action, which enable reasoning and planning of certain functionalities provided by the communication scenario in the architecture.

For simplicity only two scenarios have been formalized in this paper, the first one is an actuator device that controls AC temperature and subscribing for actuation commands, as expressed by axioms 1-6. Axioms 7-13 show that model matches subscriptions with previous registered commands and starts fetching these commands from caches and forwards them to actuators through the view. The second scenario is a user registering and publishing commands to the AC actuator, registration process is expressed by axioms 14-21 and publishing process is expressed by axioms 22-33. Actions and fluents used in these axioms are explained in Table 8 and Table 9 respectively.

Table 8. Actions Descriptions

Action	*Parameters*	*Description*
$send\_sub$	$subscriber_i, cn_j, authInfo, controller_k$	$subscriber_i$ sends *SUB* message with content name cn_j and authentication information $authInfo$ to $controller_k$.
rec_sub	$controller_k, cn_j, authInfo, subscriber_i$	$controller_k$ receives *SUB* message with content name cn_j and authentication information $authInfo$ from $subscriber_i$.
for_sub	$controller_k, cn_j, model_k$	$controller_k$ forwards cn_j to $model_k$
$check$	$controller_k, cn_j, authInfo$	$controller_k$ checks whether cn_j is a command subscription because in this case it is mandatory to authenticate the subscriber and the publishers according to the $authInfo$.
$check\_c_{nt}$	$x, y, regT$	x checks whether content name y is available in its registry table $regT$.
$checkP$	$model_k, Plc, regT$	$model_k$ checks policies in registry table
$auth$	x, y	x authenticates y.
$modfC$	$controller_k, regT$	$controller_k$ modifies its registry table $regT$ with the *SUB* information.
$modf$	$x, regT, y, cn_j, Plc$	x modifies its registry table with y identity information, and topic name cn_j and different required policies Plc such as privacy and caching

		policies.
$send\_reg$	$publisher_m, cn_j, authInfo, controller_k$	$publisher_m$ sends REG message with cn_j and $authInfo$ to $controller_k$.
rec_reg	$controller_k, cn_j, authInfo, publisher_m$	$controller_k$ receives REG message with with cn_j and $authInfo$ from $publisher_m$.
for_reg	$controller_k, cn_j, model_k$	$controller_k$ forwards REG message with cn_j to $model_k$.
$notifR$	$model_k, accept, pubId, view_k$	$model_k$ notifies $view_k$ to send acceptance message to publisher $pubId$.
$notif$	$model_k, view_k, subInfo$	$model_k$ notifies $view_k$that it will start pushing information to be sent to $subInfo$.
$send\_accept$	$view_k, publisher_m$	$view_k$ sends acceptance to $publisher_m$.
$send\_pub$	$publisher_m, ndo_{mp}, controller_k$	$publisher_m$ starts sends commands p as ndo_{mp} to $controller_k$.
rec_pub	$controller_k, nao_{mp}, puousner_m$	$controller_k$ receives ndo_{mp} from $publisher_m$.
for_pub	x, y, z	x forwards PUB message information y to z.
$fetch$	$model_k, nao_{mp}, cache_c$	$model_k$ sends fetch_message to $cache_c$ to fetch ndo_{mp}.
$prep$	$view_k, route$	$view_k$ determines and prepares route to send contents.
$push$	x, y, z	x sends $contents\ y$ to z.

Table 9. Fluents Description

Fluents	*Fluent Type*	*Description*
$require(auth, x)$	relational	X node require authentication
$authenticated(x)$	relational	X node is authenticated successfully
$cache(Plc, regT)$	functional	Returns cache address recorded in $regT$ according to a certain policy Plc
$exist(cn_j, regT)$	relational	cn_j exists in egT
$cnt(ndo_{mp})$	functional	Returns content name for ndo_{mp}

Axiom (1) implies that the situation in which a receiving SUB message action by the controller $controller_k$, has occurred as a subsequent of performing sending SUB message action by the subscriber $subscriber_i$ in previous situation s_0. The temporal order is expressed by $(s_x \prec s_y)$ in each axiom which means that s_x is the situation that precedes situation s_y. The rest of axioms 2-14 formalize the subscription and pushing available commands to new subscribers communications.

$$1\ do(rec\_sub(controller_k, cn_j, authInfo, subscriber_i), do(a, s_0)) \leftrightarrow$$

$$a = send\_sub(subscriber_i, cn_j, authInfo, controller_k)$$

2 $do(check(controller_k, cn_j, authInfo), do(a, s_1)) \leftrightarrow a = rec\_sub(controller_k, cn_j, authInfo, subscriber_i) \wedge (s_0 \prec s_1)$

3 $do(auth(controller_k, subscriber_i), do(a, s_2)) \leftrightarrow a = check(controller_k, cn_j, authInfo) \wedge (s_1 \prec s_2) \wedge require(auth, subscriber_i)$

4 $do(modfC(controller_k, regT), do(a, s_3))) \leftrightarrow a = auth(controller_k, subscriber_i) \wedge (s_2 \prec s_3) \wedge authenticated(subscriber_i)$

5 $do(for\_sub(controller_k, cn_j, model_k), do(a, s_4))) \leftrightarrow a = modfC(controller_k, regT) \wedge (s_3 \prec s_4)$

6 $do(modf(model_k, regT, subsId, cn_j, Plc), do(a, s_5)) \leftrightarrow a = for\_sub(controller_k, cn_j, model_k) \wedge (s_4 \prec s_5)$

7 $do(check\_cnt(model_k, cn_j, regT), do(a, s_6)) \leftrightarrow a = modf(model_k, regT, subsId, cn_j, Plc) \wedge (s_5 \prec s_6)$

8 $do(fetch(model_k, ndo_{mp}, cache_c), do(a, s_7)) \leftrightarrow a = check\_cnt(model_k, cn_j, regT) \wedge (s_6 \prec s_7) \wedge exist(cn_j, regT) \wedge cache(Plc, regT) = cache_c$

9 $do(notif(model_k, view_k, subInfo), do(a, s_8)) \leftrightarrow a = fetch(model_k, ndo_{mp}, cache_c) \wedge (s_7 \prec s_8)$

10 $do(push(cache_c, ndo_{mp}, model_k), do(a, s_8)) \leftrightarrow a = fetch(model_k, ndo_{mp}, cache_c) \wedge (s_7 \prec s_8)$

11 $do(prep(view_k, route), do(a, s_9)) \leftrightarrow a = notif(model_k, view_k, subInfo) \wedge (s_8 \prec s_8)$

12 $do(push(model_k, ndo_{mp}, view_k), do(a, s_9)) \leftrightarrow a = push(cache_c, ndo_{mp}, model_k) \wedge (s_8 \prec s_9)$

13 $do(push(view_k, ndo_{mp}, subscriber_i), do(a, s_{10})) \leftrightarrow a = push(model_k, ndo_{mp}, view_k) \wedge (s_9 \prec s_{10})$

Axioms 14 -33 express the details of the registration, publishing, and pushing commands to previously subscribed subscribers communications.

14 $do(rec\_reg(controller_k, cn_j, authInfo, publisher_m), do(a, s_0)) \leftrightarrow$

$$a = send\_reg(publisher_m, cn_j, authInfo, controller_k)$$

15 $do(check\_cnt(controller_k, cn_j, regT), do(a, s_1)) \leftrightarrow a = rec\_reg(controller_k, cn_j, authInfo, publisher_m) \wedge (s_0 \prec s_1)$

16 $do(auth(controller_k, publisher_m), do(a, s_2)) \leftrightarrow a = check\_cnt(controller_k, cn_j, regT) \wedge (s_1 \prec s_2) \wedge exist(cn_j, regT) \wedge require(auth, publisher_m)$

17 $do(modfC(controller_k, regT), do(a, s_3))) \leftrightarrow a = auth(controller_k, publisher_m) \wedge (s_2 < s_3) \wedge authenticated(publisher_m)$

18 $do(for\_reg(controller_k, cn_j, model_k), do(a, s_4))) \leftrightarrow a = modfC(controller_k, regT) \wedge (s_3 < s_4)$

19 $do(modf(model_k, regT, pubId, cn_j, Plc), do(a, s_5)) \leftrightarrow a = for\_reg(controller_k, cn_j, model_k) \wedge (s_4 < s_5)$

20 $do(notifR(model_k, accept, pubId, view_k), do(a, s_6)) \leftrightarrow a = modf(model_k, regT, pubId, cn_j, Plc) \wedge (s_5 < s_6)$

21 $do(send\_accept(view_k, publisher_m), do(a, s_7)) \leftrightarrow a = notifR(model_k, accept, pubId, view_k) \wedge (s_6 < s_7)$

22 $do(send\_pub(publisher_m, ndo_{mp}, controller_k), do(a, s_8)) \leftrightarrow a = send\_accept(view_k, publisher_m) \wedge (s_7 < s_8)$

23 $do(rec\_pub(controller_k, ndo_{mp}, publisher_m), do(a, s_9)) \leftrightarrow a = send\_pub(publisher_m, ndo_{mp}, controller_k) \wedge (s_8 < s_9)$

24 $do(check\_cnt(controller_k, cnt(ndo_{mp}), regT), do(a, s_{10})) \leftrightarrow a = rec\_pub(controller_k, ndo_{mp}, publisher_m) \wedge (s_9 < s_{10})$

25 $do(for\_pub(controller_k, ndo_{mp}, model_k), do(a, s_{11})) \leftrightarrow a = check\_cnt(controller_k, cnt(ndo_{mp}), regT) \wedge (s_{10} < s_{11}) \wedge exist(cnt(ndo_{mp}), regT)$

26 $do(checkP(model_k, Plc, regT), do(a, s_{12})) \leftrightarrow a = for\_pub(controller_k, ndo_{mp}, model_k) \wedge (s_{11} < s_{12})$

27 $do(for\_pub(model_k, ndo_{mp}, cache_c), do(a, s_{13})) \leftrightarrow a = checkP(model_k, Plc, regT) \wedge (s_{12} < s_{13}) \wedge cache(Plc, regT) = cache_c$

28 $do(check\_cnt(model_k, cnt(ndo_{mp}), regT), do(a, s_{14})) \leftrightarrow a = for\_pub(model_k, ndo_{mp}, cache_c) \wedge (s_{13} < s_{14})$

29 $do(notif(model_k, view_k, subInfo), do(a, s_{15})) \leftrightarrow a = check\_cnt(model_k, cnt(ndo_{mp}), regT) \wedge (s_{14} < s_{15}) \wedge exist(cnt(ndo_{mp}), regT) \wedge cache(Plc, regT) = cache_c$

30 $do(push(model_k, ndo_{mp}, view_k), do(a, s_{16})) \leftrightarrow a = notif(model_k, view_k, subInfo) \wedge (s_{15} < s_{16})$

31 $do(prep(view_k, route), do(a, s_{16})) \leftrightarrow a = notif(model_k, view_k, subInfo) \wedge (s_{15} < s_{16})$

$$32\ do(push(view_k, ndo_{mp}, subscriber_i), do(a, s_{17})) \leftrightarrow a = push(model_k, ndo_{mp}, view_k) \wedge (s_{16} \prec s_{17})$$

Reasoning using Situation Calculus**:**

The following is a reasoning example to proof that no publisher can publish data without being authenticated. Assuming that from situation s=22, $send\_pub(publisher_m, ndo_{mp}, controller_k)$, and having the fact that $\neg authenticated(publisher_m)$. Going back in situations we should reach a situation where a publisher is authenticated in a time or situation before publishing and $\neg authenticated(publisher_m)$ is false:

do($send\_pub(publisher_m, ndo_{mp}, controller_k)$,do($send\_accept(view_k, publisher_m)$,

do($notifR(model_k, accept, pubId, view_k)$,do($modf(model_k, regT, pubId, cn_j, Plc)$,

do($for\_reg(controller_k, cn_j, model_k)$,do($modfC(controller_k, regT)$,

do($auth(controller_k, publisher_m), s2$)))))))

As can be seen from the example, going back in situations from situation s22 where publishers sends data using action *send-pub*, we reached action *auth* in situation s2, which implies that publishers can only publish data if in some previous situation they were authenticated. Moreover, in situation s3, action $modfC(controller_k, regT)$ occurred after the occurrence of $auth(controller_k, publisher_m)$ in situation s2 and $authenticated(publisher_m)$ is true, which leads into a conflict according to the start assumption of $\neg authenticated(publisher_m)$, which makes us conclude that assumption $\neg authenticated(publisher_m)$ is false.

Planning using Situation Calculus:

The following is an example of planning for pushing *NDOs* to subscribers who subscribed for certain topic cn_j in GoLog language:

$Proc\ Push\_Data$

$check\_cnt(model_k, cn_j, regT)$

$while\ \exists r.cnt(r) = cn_j, r = ndo$

$fetch(model_k, ndo_{mp}, cache_c)$

$notif(model_k, view_k, subInfo)$

$push(cache_c, ndo_{mp}, model_k)$

$push(model_k, ndo_{mp}, view_k)$

$push(view_k, ndo_{mp}, subscriber_i)$

End while

End Proc Push_Data

5. Conclusion

Future network is crowded with tremendous number of different types of devices, which vary in their capabilities and characteristics. These uniquely identified devices are connected through the internet to form the IoT. New networking architectures are investigated as an alternate to current IP network architecture to handle IoT challenges such as mobility, real-time communication, constrained devices, and the large amount of generated data. One of these promising paradigms is ICN.

This paper proposes a privacy-aware IoT architecture based on ICN paradigm, the communication pattern adopted in this proposed architecture is by using MVC-Pub/Sub architecture rather than classical N-Tier architecture. The proposed architecture inherited ICN, MVC and Pub/Sub advantages. ICN in-network caching and retrieving data by its name rather than IP addresses enhance network efficiency, and mobility. Pub/Sub approach decouples subscribers and publishers in location and space, which supports both publisher's and subscriber's mobility, in addition to supporting their privacy. The MVC communication pattern enhances network efficiency, added additional layers of abstraction between the architecture components to enhance interoperability, and facilitates deployment choices. The overall architecture supports privacy in different aspects such as privacy of data, privacy of user's identities, and location privacy.

A comparison of overall communications and processing cost between N-Tier and MVC architectures based on full publish/subscribe scenario shows that MVC outperforms N-Tier by dividing the architecture tasks over its components, rather than overloading controllers or Business Logic components. Parallelism supported by MVC also enhances the overall performance. Communication cost is analyzed for request/response, and Pub/Sub approaches. Moreover, various caching techniques for IoT periodic data is modeled to calculate the hits probabilities, taking into considerations freshness, request frequency, and cache size. Reasoning and planning for publishing actuation commands scenarios is performed using Situation Calculus which is used to formalize the communications in causality and temporal framework.

Future researches will be dedicated on investigating and implementing privacy techniques in the mentioned aspects, such as private routing techniques and light data encryption mechanisms.

References

Ahlgren, B. et al. (2012). A survey of information-centric networking. *IEEE Communications Magazine, 50*(7), pp. 26-36.

Almobaideen, W., Krayshan, R., Allan, M., & Saadeh, M. (2017). Internet of Things: Geographical Routing based on healthcare centers vicinity for mobile smart tourism destination. *Technological Forecasting and Social Change, 123*(October 2017), 342-350.

Amadeo, M. et al. (2016). Information-centric networking for the internet of things: challenges and opportunities. *IEEE Network, 30*(2), 92-100.

Amadeo, M. et al. (2017). *A Cloud of Things framework for smart home services based on Information Centric Networking.* Calabria, Italy, s.n.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks, 54*(15), 2787–2805.

Bari, M. F. et al. (2012). A Survey of Naming and Routing in Information Centric-Networks. *IEEE Communications Magazine, 50*(12), 44-53.

Carlos-Mancilla, M., López-Mellado, E., & Siller, a. M. (2016). Wireless Sensor Networks Formation: Approaches and Techniques. *Journal of Sensors, 2016.*

Chai, W. K. et al. (2011). CURLING: Content-Ubiquitous Resolution and Delivery Infrastructure for Next-Generation Services. *IEEE Communications Magazine, 49*(3), 112-120.

Chen, J. et al. (2016). Exploiting ICN for Realizing Service-Oriented Communication in IoT. *IEEE Communications Magazine, 54*(12), 24 - 30.

Dannewitz, C. et al. (2013). Network of Information (NetInf) – An information-centric networking Architecture.

Computer Communications, 36(7), 721-735.

Detti, A., Melazzi, N. B., Salsano, S., & Pomposini, M. (2011). *CONET: a content centric inter-networking architecture.* Toronto, Ontario, Canada, s.n.

Fielding, R. T. & Taylor, R. N., 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT), 2*(2), 115-150.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* s.l.:UNIVERSITY OF CALIFORNIA.

Fotiou, N., Nikander, P., Trossen, D., & Polyzos, G. C. (2010). *Developing Information Networking Further:From PSIRP to PURSUIT.* Athens, Greece, s.n.

Ghodsi, A. et al. (2011). *Naming in content-oriented architectures.* Toronto, Ontario, Canada, s.n.

GuangChun, L., WangYanhua, Xianliang, L., & Hanhong, (2003). A Novel Web Application Frame Developed by MVC. *Software Engineering Notes, 28*(2), 1-3.

Happ, D., Karowski, N., Menzel, T. & Handziski, V., 2017. Meeting IoT platform requirements with open pub/sub solutions. *Annals of Telecommunications, 72*(1-2), 41-52.

Heimgaertner, F. et al. (2015). *A security architecture for the publish/subscribe C-DAX middleware.* London, UK, s.n.

Hoefling, M. et al. (2015). *Enabling resilient smart grid communication over the information-centric C-DAX middleware.* Cottbus, Germany, s.n.

Hoefling, M., Heimgaertner, F., & Menth, M. (2016). *Advanced communication modes for the publish/subscribe C-DAX middleware.* Istanbul, Turkey, s.n.

Ioannou, A., & Weber, S. (2016). A Survey of Caching Policies and Forwarding Mechanisms in Information-Centric Networking. *IEEE Communications Surveys & Tutorials, 18*(4), 2847-2886.

Jacobson, V. et al. (2009). *Networking Named Content.* Rome, Italy, s.n.

Jin, H., Xu, D., Zhao, C., & Liang, D. (2017). Information-Centric mobile caching network frameworks and caching optimization: a survey. *EURASIP Journal on Wireless Communications and Networking,* pp. 1-32.

Kim, K. et al. (2015). An interactive pervasive whiteboard based on MVC architecture for ubiquitous collaboration. *Multimedia Tools and Applications, 74*(5), 1557-1576.

Koponen, T. et al. (2007). *A data-oriented (and beyond) network architecture.* Kyoto, Japan, s.n.

Kraijak, S., & Tuwanut, P. (2015). *A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends.* Shanghai, s.n.

Levesque, H., Pirri, F., & Reiter, R. (1998). Foundations for the Situation Calculus. *Link oping Electronic Articles in Computer and Information Science, 3*(18), 1-18.

Li, S., Xu, L. D., & Zhao, S. (2014). The internet of things: a survey. *Information Systems Frontiers, 17*(2), 243-259.

Li, S., Xu, L. D., & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers, 17*(2), 243-259.

Li, S., Zhang, Y., Raychaudhuri, D., & Ravindran, R. (2014). *A Comparative Study of MobilityFirst and NDN based ICN-IoT Architectures.* Rhodes, IEEE.

Lindgren, A. et al. (2014). *Applicability and Tradeoffs of Information-Centric Networking for Efficient IoT draft-lindgren-icnrg-efficientiot-00.* Retrieved April 25, 2016, from https://tools.ietf.org/html/draft-lindgren-icnrg-efficientiot-00

Mamun, Q. (2012). A Qualitative Comparison of Different Logical Topologies for Wireless Sensor Networks. *Sensors, 12*.

Melazzi, N. B. (2011). *CONVERGENCE: extending the media concept to include representations of real world.* Retrieved November 10, 2017, from http://www.ict-convergence.eu/wp-content/uploads/C1.pdf

Pavlou, G., Wang, N., Chai, W. K., & Psaras, I. (2013). Internet-scale content mediation in information-centric networks. *annals of telecommunications, 68*(3-4), 167-177.

Quevedo, J., Corujo, D., & Aguiar, R. (2014). *A case for ICN usage in IoT environments.* Austin, s.n.

Reenskaug, T. (1979). *The original MVC reports ,* Oslo: Dept. of Informatics , University of Oslo.

Saadeh, H., Almobaideen, W. & Sabri, K. E., 2017. *Internet of Things: A review to support IoT architecture's design.* Amman, Jordan, s.n.

Salsano, S. et al. (2011). *System Architecture D3.2.* Retrieved Novmber 5, 2017, from http://www.ict-convergence.eu/deliverables/

Seskar, I., Nagaraja, K., Nelson, S., & Raychaudhuri, D. (2011). *MobilityFirst future internet architecture project.* Bangkok, Thailand, s.n.

SHENG, Z. et al. (2013). A SURVEY ON THE IETF PROTOCOL SUITE FOR THE INTERNET OF THINGS: STANDARDS, CHALLENGES, AND OPPORTUNITIES. *IEEE Wireless Communications ,* 20(6), 91-98.

Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2017). *A secure ICN-IoT architecture.* Paris, France, s.n.

Solapure, S. S., & Kenchannavar, H. (2016). *Internet of Things: A survey related to various recent Architectures and Platforms available.* Jaipur, s.n.

Suarez, J. et al. (2016). A secure IoT management architecture based on Information-Centric Networking. *Journal of Network and Computer Applications, 63*, 190-204.

TAGAMI, A. & ARUMAITHURAI, M., 2016. GreenICN Project: Architecture and Applications of Green Information Centric Networking. *IEICE Transactions on Communications,* E99.B(12), 2470-2476.

Xiaoke, J., Jun, B., Guoshun, N., & Zhaogeng, L. (2015). A Survey on Information-Centric Networking: Rationales, Designs and Debates. *China Communications, 12*(7), 1-12.

Xylomenos, G. et al., 2014. A Survey of Information-Centric Networking Research. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, 16*(2), 1024-1049.

Yoo, S., Son, J. H. & Kim, M. H., 2009. A scalable publish/subscribe system for large mobile ad hoc networks. *The Journal of Systems and Software, 82*(7), 1152-1162.

Yue, H. et al., 2014. DataClouds: Enabling Community-Based Data-Centric Services Over the Internet of Things. *IEEE Internet of Things Journal, 1*(5), 472 - 482.