A Secure Network Architecture for the Internet of Things Based on Local Authorization Entities

Hokeun Kim, Armin Wasicek, Benjamin Mehne and Edward A. Lee
Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley
Email: hokeunkim@eecs.berkeley.edu, arminw@berkeley.edu, bmehne@eecs.berkeley.edu, eal@eecs.berkeley.edu

***Abstract*—Security is essential to enable the Internet of Things (IoT). Key security measures that work well on the traditional Internet, however, do not necessarily adapt well to the IoT. Specifically, authentication and/or authorization based on certificates provided by certificate authorities (CAs) cannot, in current form, scale to the expected 50 billion devices. And widely used encryption technologies for the Internet require too much energy for resource-constrained devices. This paper describes a secure network architecture with key distribution mechanisms using local, automated authorization entities. The architecture provides security guarantees while addressing IoT-related issues including resource constraints. For evaluation, we show that the architecture's overhead scales at a significantly slower rate than widely used SSL/TLS and works well with resource-constrained devices.**

***Keywords*—Internet of Things; Network security; Key management; Authentication; Authorization**

I. Introduction

The Internet of Things (IoT) [1] faces challenges [2] to enable scalable, safe and secure systems, possibly with resource constraints. Since the IoT interacts with humans, machines and environments, failures in the IoT can lead to very serious consequences. This fact makes safety of the IoT particularly important. Safety extends to security in the sense that security guarantees (e.g., protection from intrusion or unauthorized access) can help prevent an adversary from damaging safety. Safety measures such as Airbus flight envelope protection [3], which prohibits pilots from performing risky maneuvers, can help prevent a successful intruder from doing damage.

The security of the traditional Internet has been enhanced by well-developed security measures such as the SSL/TLS (Secure Socket Layer / Transport Layer Security) protocol suites[1]. However, the IoT has unique characteristics that distinguish it from the traditional Internet, and these characteristics lead to special requirements for a secure network architecture of the IoT. Widely used network security measures do not adapt one-to-one to the IoT because of these special requirements.

For example, IoT components including electric vehicles (EVs) and EV charging infrastructure in Fig. 1 are considered safety-critical. Unlike servers in data centers, EVs and EV charging stations are physically accessible not only by valid users but also by potential adversaries. This leads to the increased number of physical points of access, thus, there can be a higher risk of being subverted. Therefore, the secure network architecture for the IoT must have ways to revoke authorization of the devices within a short amount of time to limit the damage when they are under control of adversaries.

[1]Widely used by web servers, clients, and remote logins.

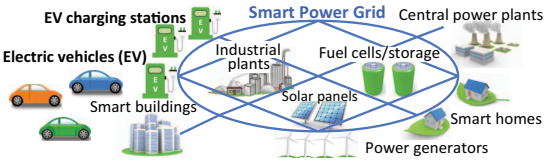

Fig. 1. Electric vehicles (EVs) and charging infrastructure

In addition, mobile phones or EVs can migrate from one network to another, possibly making network connection unstable. There are also IoT devices with constrained resources and the number of IoT devices is expected to grow rapidly. Therefore, the secure network architecture should work well with unstable connection and resource-constrained devices at a great scale. However, with security measures such as TLS based on certificates provided by certificate authorities, it will be very difficult to have control over authorization of a huge number of devices, possibly with resource constraints.

We also claim that the secure network architecture for the IoT should be able to provide security guarantees at a comparable level as TLS, at least for some devices, for safety. Therefore, it is not enough to simply adapt lightweight security solutions for wireless sensor networks (WSNs) that make tradeoffs in terms of security guarantees.

In this paper, we propose a secure network architecture using local, automated authorization entities to address the IoT-related requirements while providing high-level security guarantees. The proposed network architecture provides key management mechanisms that scale very well and work well with devices with resource constraints or unstable connections. The proposed architecture can use the cryptography algorithms that are used in TLS to provide a comparable level of security guarantees. We also carry out experiments that evaluate the effectiveness of our approach, and compare with TLS.

The organization of the paper is as follows: Section II describes the security requirements for the IoT, and is followed by a review of current network security measures in Section III. Section IV describes the proposed approach, which is evaluated in Section V. Section VI concludes.

II. IoT Security Requirements

This section summarizes security requirements of the secure network architecture and key management systems for the IoT.

978-1-5090-4052-0/16 $31.00 © 2016 IEEE
DOI 10.1109/FiCloud.2016.24

- **Frequent authorization and authentication** – Due to the criticality of some devices in the IoT, tight authorization and authentication will be necessary. Machines operate at higher speeds than human beings, and physical accessibility of devices creates more vulnerability. Dynamically varying situations resulting from mobility may change authorization of devices. Moreover, frequent authorization can provide ways to limit the damage in case critical devices are subverted.
- **Automated mutual authentication** – It will be prohibitive for users to remember passwords for a large number of devices. Thus, the IoT devices must be able to authenticate themselves without user intervention.
- **Intermittent connectivity** – Mobility of devices changes the network environment under which the devices operate, which can lead to unstable connectivity. However, we cannot compromise security when network connection is unstable, thus we should be able to handle intermittent connectivity of devices.
- **Dynamic entity registration** – Unlike the traditional Internet, the IoT includes devices with shorter life cycles than general computers. Mobile devices may be added to and removed from authentication/authorization systems dynamically. Therefore, adding and removing entities should be manageable in the secure network architecture for the IoT.
- **Support for scalability features** – There are a variety of approaches for network scalability, which benefit the IoT devices, including the publish-subscribe protocols [4] such as MQTT [5]. The security architecture should be able to work together with these scalability features.
- **Consideration for resource constraints** – Some devices in the IoT have limited resources; for example, battery-powered devices have limited energy budget for computation/communication. Stronger security measures generally cost more energy; however, excessive energy consumption can harm the availability. Hence, the security measure should be able to balance security and energy consumption.
- **Privacy** – Personal devices such as mobile phones can easily collect and carry private information of their users. Therefore, the security measure also needs to be able to protect user's privacy.

III. Security Measures In the Field

Secure Socket Layer / Transport Layer Security (SSL/TLS), or simply TLS [6], has been providing security for the Internet. For authentication, TLS uses certificates, usually provided by a certificate authority (CA). However, this cannot be the best option for the IoT due to the overhead for CAs to manage the huge number of certificates. To make TLS with CAs more scalable, the Let's Encrypt project[2] launches free and automated CAs. Nevertheless, it will be too demanding for resource-constrained devices to carry large certificates and perform computationally expensive asymmetric key (public key cryptography) operations for every TLS connection.

[2]https://letsencrypt.org

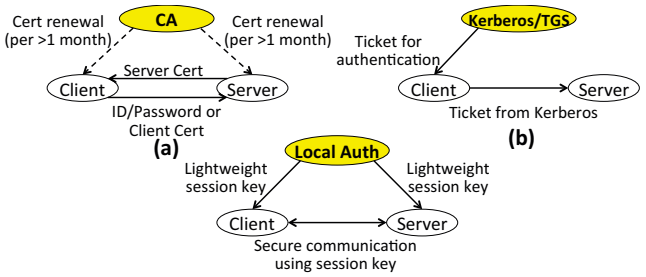

Fig. 2. Authentication/authorization flows of different approaches; (a) CA (certificate authority) and Certs (certificates) (b) Kerberos (authentication service) / TGS (ticket granting service) (c) Proposed local Auth (Authentication/Authorzation entity)

Kothmayr *et al.* [7] propose an authentication system for the IoT using DTLS [8], a datagram variant of TLS. The constrained application protocol (CoAP) [9] is secured by DTLS. This kind of approach can work better with resource-constrained devices than TLS. However, DTLS is still based on point-to-point connections like TLS, which makes it challenging to secure one-to-many communications common in the IoT such as broadcasting or publish-subscribe patterns [4]. In addition, a certificate used in TLS and DTLS contains a unique value for an entity. This risks exposing the entity's identity, leading to potential threat to privacy.

Another issue with certificate-based approaches is revocation of authentication. As illustrated in Fig. 2 (a), the CA is only involved in the issuance of certificates, and the client and server authenticate each other using the certificates as long as the certificates are valid. The validity period of certificates is typically longer than several months due to the management overhead of certificates, especially for renewal of certificates. Therefore, it is challenging to revoke authentication of entities using certificates. This can be potential threat to safety of critical components in the IoT in case they are compromised.

For authentication of entities, the Kerberos authentication system [10] issues temporary tickets through its ticket granting service (TGS), as illustrated in Fig. 2 (b). Thus, Kerberos provides a centralized control over the validity period of authentication, addressing the challenges of revocation. However, such systems are designed for human users, requiring user intervention such as entering passwords. This makes it hard to support automated mutual authentication of IoT devices.

There are extensions for Kerberos [11] that use public key cryptography for authentication, to replace the human intervention. Even with these extensions, however, clients with intermittent connectivity may face authentication problems when they are not connected to Kerberos/TGS. Although caching tickets can address the intermittent connectivity problem, it creates another problem of allowing a compromised client with cached tickets to authenticate with the server. This is because a ticket is delivered to a server by a client, not by Kerberos/TGS, as shown in Fig. 2 (b), and the server trusts the client as long as the ticket is valid.

The authentication flows for certificate-based approaches

and the Kerberos authentication system are designed for the Internet with general-purpose computers. Although these approaches have been successful for the traditional Internet, we note that the authentication flows shown in Fig. 2 (a) and (b) cannot address some of the IoT-related security and scalability issues. Therefore, we propose a network architecture that has the authentication/authorization flow as shown in Fig. 2 (c).

In the proposed network architecture, a local authorization entity, *Auth*, assigns lightweight session keys to entities involved in communication. The Auth entity controls the validity period of session keys and covers authorization as well as authentication. This is possible because Auth is aware of communication context of registered entities, determining whether an entity is authorized to communicate with others. While Auth serves as a local point of authorization, it also interacts with other Auths to control communication between entities registered with different Auths, distributing the authorization overhead locally. Potential deployment targets for Auth include Intel's IoT gateways[3] and the SwarmBox[4] from the TerraSwarm project (https://www.terraswarm.org/).

Security measures for wireless sensor networks (WSN) support automated authentication for resource-constrained devices. Approaches using random or pairwise pre-distributed encryption keys [12], [13] provide energy efficient solutions. There has been a static key management approach that allows key establishment for newly added nodes [14]. However, static key management systems may not be proper for the safety-critical devices running under hostile environments, due to the increased probability of being attacked for the cryptographic key with a long lifetime.

He *et al.* [15] survey dynamic key management systems for WSN, which can address the problem with pre-distributed keys by supporting key revocation mechanisms. These approaches for WSN can be categorized into *distributed* and *centralized approaches*. In *distributed approaches* including EDDK [16] (Energy-efficient Distributed Deterministic Key management), neighboring nodes collaborate to dynamically establish keys. Such approaches can avoid a single point of failure in authentication systems; however, they tend to be more vulnerable to collusion attacks and prone to design errors.

In *centralized approaches*, a central trusted third party (e.g., a base station) is responsible for key generation and distribution for nodes. Many of these approaches have similar authentication flows as the proposed approach's authentication flow in Fig. 2 (c). Huang *et al.* [17] propose a centralized forward authentication approach for hierarchical and heterogeneous sensor networks composed of high-end and low-end sensor nodes. Sahingoz [18] provides a key distribution system using an unmanned aerial vehicle (UAV) as a center of key distribution and coordination, for large scale WSNs. To support addition and deletion of mobile sensor nodes, Erfani *et al.* [19] propose a key management system that uses key pre-distribution and post-deployment key establishment.

[3]http://www.intel.com/content/www/us/en/internet-of-things/gateway-solutions.html

[4]https://swarmlab.eecs.berkeley.edu/projects/5378/swarmbox

The dynamic key management systems for WSN can address some part of the IoT-related security requirements in Section II. However, they still have limited results to cover the great heterogeneity of devices in the IoT, from the resource-constrained sensor nodes to the critical components that require frequent authentication and authorization for safety. The support for the dynamic environment such as intermittent connectivity and dynamic entity registration is still not sufficient.

IV. Proposed Approach

Our proposed network architecture uses a local authorization entity, which we call *Auth*, to handle frequent authentication and authorization of devices. For scalability and resource constraints, our approach uses symmetric keys for authentication and authorization rather than asymmetric keys. Before diving into the details of our approach, we clarify some important terms used throughout this paper.

- *Entity* – Any device connected to the network in the IoT to be authenticated and authorized. Each entity has a unique identifier and cryptographic keys for secure communication.
- *Auth* – An entity responsible for authenticating and authorizing registered entities. Auth maintains and manages database tables to store information of entities.
- *Client* – An entity that initiates communication.
- *Server* – An entity accepting communication requests.
- *Public key* and *Private key* – The public and private components of entity's asymmetric key pair.
- *Distribution key* – A symmetric key-wrapping key used to encrypt session keys for distribution.
- *Session key* – A symmetric key used to protect a single session of communication. Since only authorized entities receive a valid session key, an entity can prove that it is authorized, by proving ownership of the session key. Each session key is assigned a unique ID and validity periods.

The operations of the proposed approach can be divided into four phases, (1) *entity registration*, (2) *session key distribution*, (3) *communication initialization*, and (4) *secure communication*, as shown in Fig. 3. A newly added entity must be registered with at least one Auth during the entity registration phase. Registered entities can obtain session keys through session key distribution. The dotted lines below the client and server describe the time line of communication initialization and secure communication. The following sections explain the roles of Auth and details of each phase.

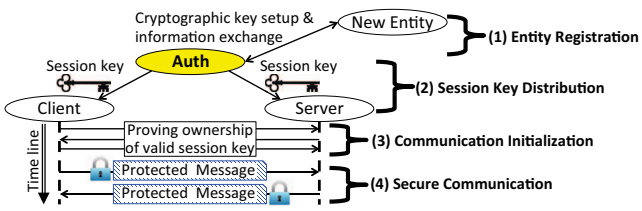

Fig. 3. Overview of four phases of the proposed approach

(a) Registered Entity Table (Dist.: Distribution, Algo.: Algorithm, Prec.: Precision)

Name	Group	Public key	Dist. Key <Value, Expires on>	Key Dist. Conditions			Operational Conditions
				Cipher Algo.	Hash Algo.	Dist. Key Validity	
EV001	Electric Vehicle	8c 98 7c ...	<28 3d ef ..., 9/1/16,16:00>	AES-128 -CBC	SHA 256	10 days	Max session keys:30, Session keys/req:5, ...
CS003	EV Charging Station	30 8b 3b ...	<da 9f f1 ..., 8/23/16,4:00>	AES-256 -CBC	SHA 384	12 hours	Max session keys:5, Time prec.: 100us,...
FC005	Fuel Cell Storage	2c 4a b1 ...	<d3 28 3d ..., 9/21/16,16:00>	AES-128 -CBC	SHA 384	30 days	Max session keys:10, Session keys/req:10,...
CS024	Current Sensor	N/A	<ef c0 20 ..., 8/22/18,16:00>	AES-128 -CBC	SHA 256	2 years	Max session keys: 10, Session keys/req: 5,...
...	...	...	...	...			...

(b) Communication Policy Table

Client Entity (Initiator)	Server Entity (Listener)	Cipher Algorithm	Hash Algorithm	Session Key Validity Period <Absolute, Relative>
Electric Vehicle Group	EV Charging Station Group	AES-128-CBC	SHA256	<2 days, 30 min>
EV Charging Station Group	Fuel Cell Storage Group	AES-256-GCM	SHA384	<1 hour, 5 min>
Fuel Cell Storage Group	Current Sensor Group	RC4-128	SHA1	<7 days, 1 day>
...	...	...	...	...

(c) Cached Session Key Table

Session Key ID	Session Key Value	Cipher Algorithm	Hash Algorithm	Owner(s)	Expires on (Absolute)	Valid for (Relative)
212371	42 d5 49 1c ...	AES-128-CBC	SHA256	EV001	8/24/2016, 16:00	30 min
212372	0a de d6 90 ...	AES-128-CBC	SHA256	EV001	8/24/2016, 16:00	30 min
212373	56 e2 f8 1a ...	AES-256-GCM	SHA384	FC005	8/22/2016, 17:00	5 min
212374	37 6b 1a b9 ...	RC4-128	SHA1	CS024	8/29/2016, 16:00	1 day
...	...	...	...	...	...	...

Fig. 4. Examples of Auth's database tables

A. Auth

The Auth entity allows authentication and authorization through distributing session keys to entities that are valid for communication. For session key distribution, Auth stores various information in its database. Fig. 4 illustrates examples of database tables maintained and managed by Auth.

1) Registered entity table: The registered entity table in Fig. 4 (a) stores entity-specific information such as an entity's unique name, group, public/distribution key, and conditions for key distribution and operations. The key distribution conditions determine the cipher/hash algorithm and validity period for the distribution key. The distribution key field has the key's value and when it expires. The validity period or cryptoperiod of cryptographic keys including the distribution key can be determined depending on risk factors as recommended in [20]. Risk factors include the cryptography strength, operating environment, number of transactions, and potential threats.

In the example table Fig. 4 (a), an EV charging station named CS003, has the shortest validity period for distribution keys. This is because the charging station operates in an open space, requires a relatively large number of session keys to interact with many vehicles, and causes critical damage to the power grid if compromised. An electric vehicle named EV001 has a longer validity period because it interacts with others less frequently and it is less accessible (e.g., protected by locks/garage doors). A fuel cell storage, FC005 has an even longer cryptoperiod, since it operates under restricted access.

The distribution key can be updated using public key cryptography, thus, Auth stores public keys of the registered entities as in Fig. 4 (a). The proposed approach also supports devices incapable of public key cryptography such as the (electric) current sensor named CS024. In this case, Auth stores distribution key during the entity registration phase and uses it for the entire life cycle of the device. Sections IV-B and IV-C explain the details of this. The operational conditions include the maximum number of total cached session keys, a maximum number of session keys per request, and the entity's time precision required for time synchronization.

2) Communication policy table: The communication policy table in Fig. 4 (b) is used to determine the conditions of communication between certain entities such as cipher/hash algorithms and validity periods. A client (or communication initiator) and a server (or connection listener) can also be specified as a group to set a communication policy between groups of entities. With this table, Auth has a full control over authentication/authorization of entities, and the communication policies can be dynamically managed.

The cipher/hash algorithms used for communication can be determined by various criteria, including security requirements (e.g., confidentiality, integrity), available resources of entities, and performance/costs of algorithms on entities. A session key validity period is a tuple of two values, *absolute validity period* and *relative validity period*. The absolute validity period specifies how long a session key is valid after creation and the relative validity period denotes the validity of the session key after its first use for communication.

3) Cached session key table: Once a session key is generated, Auth stores it in the cached session key table in Fig. 4 (c). Auth assigns each session key a unique identifier, called session key ID. When a session key is distributed to entities, Auth updates its owners, the entities who share the session key. The *Expires on* and *Valid for* fields of each session key are set according to the communication policy table.

4) Scalability and robustness of Auth: The proposed secure network architecture supports multiple Auths, making the approach more scalable. When entities registered with different Auths want to communicate, each entity just needs to contact with its own Auth for authorization. Auths communicate with one another to deliver the same session key to their entities for setting up a secure connection. This process is explained in detail with an example in Section IV-F.

We note that Auth is a logical entity that can be implemented in a variety of ways, like the controller of a software-defined network (SDN). It is a logical entity and can be implemented in a distributed way [21]. We expect that Auth can also be implemented in a distributed manner, possibly with replicas. Such Auth implementations can avoid being a single point of failure, providing robustness against denial of service attacks. We also note that Auth must be in a safe place where only valid users can access, like general servers in data centers.

B. Entity Registration Phase

In the entity registration phase, Auth and a newly added entity exchange information for session key distribution. There are two possible options for the distribution key, *updatable distribution key* and *permanent distribution key*, depending on whether the distribution key can be updated using public key cryptography. Fig. 5 shows types of exchanged data during the entity registration, and their security requirements.

When the distribution key of the new entity is *updatable*, Auth and the entity should exchange their public keys for

Distribution Key Options	Updatable Distribution Key	Permanent Distribution Key
Key Setup Information	Auth's Public Key, Entity's Public Key	Symmetric Distribution Key
Additional Information	Auth's ID, Network Address (or URL), Entity's Unique Name, Group,	

Data Security Requirements: Integrity; Integrity + Confidentiality

Fig. 5. Exchanged information during entity registration phase, and data security requirements for different types of data

the secure delivery of the distribution key from Auth to the entity. The public keys must not be tampered with during the exchange to prevent an adversary from spoofing identity of Auth or the new entity, although the public keys need not to be confidential. If the distribution key of the new entity is *permanent*, meaning that a single distribution key is to be used for the entire life cycle of the entity, the distribution key must be kept confidential, only known to Auth and the entity.

For both options, Auth and the new entity should exchange additional information, including the new entity's unique name and group, and the Auth's ID and network address or URL (can be more than one). Data integrity must be guaranteed for the additional information exchanged to prevent masquerading. After the entity registration, Auth stores the entity's information in its registered entity table. The new entity stores Auth's information in its local storage. If the Auth is replicated, then the entity stores information of Auth's replicas as well.

As long as the specified data security requirements are satisfied, many methods can be used for the entity registration. An authorized person can plug a new device or use near field communication (NFC) to securely connect to Auth or Auth's delegate that is connected to Auth via a secure connection, for information exchange. For personal devices, for example, two-factor authentication [22] can be used to guarantee authenticity of the information from the device.

C. Session Key Distribution Phase

Auth delivers one or more session keys to an entity during the session key distribution phase. An entity can request multiple session keys a priori for future use. Hence, an entity with intermittent connectivity can authenticate and authorize itself even without direct connection to Auth. A resource-constrained entity can save communication and computation to obtain session keys. Session key distribution works on top of TCP/IP to ensure reliable message delivery. Fig. 6 illustrates the session key distribution process for two cases, depending on whether there is an available distribution key.

When an entity already has a valid distribution key as in Fig. 6 (a), the distribution key is used for the session key distribution. When a TCP/IP connection is established, Auth sends AUTH_HELLO, including Auth's information such as its ID, and a nonce (random number) generated by Auth. Upon receiving AUTH_HELLO, the entity sends SESSION_KEY_REQ, which specifies the requesting entity's name, communication purpose including the target of communication, the number of requested keys, and the session key's identifier, if necessary. The nonce from Auth and the entity's own nonce are appended to SESSION_KEY_REQ, to prevent

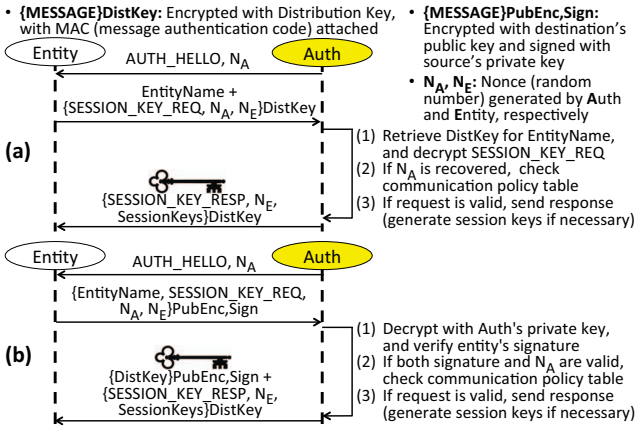

Fig. 6. Two cases of session key distribution phase (a) when the distribution key is available, and (b) when the distribution key needs to be updated.

replay attacks. The whole message is encrypted with the distribution key after attaching MAC (message authentication code). The entity's unique name is also sent in plain text.

Upon receiving the entity's request, Auth retrieves the distribution key using the entity's name, checks the Auth's nonce and the communication policy table for the entity's eligibility, and sends the response if the request is valid. Auth generates session keys before sending the response, if it is necessary. The Auth's response, SESSION_KEY_RESP includes the entity's nonce, communication policy and session keys, encrypted by the distribution key. After receiving SESSION_KEY_RESP, the entity checks the nonce and stores the session keys.

Fig. 6 (b) describes the case when the distribution key should be updated, after registering a new entity or expiry of a distribution key. In this case, public key cryptography is used to deliver the distribution key securely. The entity's request, SESSION_KEY_REQ along with the entity's name and the nonces should be encrypted with Auth's public key and signed with the entity's private key. When Auth receives the request, it decrypts the request with its private key and verifies the signature using the entity's public key. If the signature and the nonce are valid, Auth checks the request's eligibility and responds. The response includes the distribution key, *DistKey* encrypted with the entity's public key and signed with Auth's private key. The response also contains SESSION_KEY_RESP, a list of session keys and nonces, encrypted with DistKey.

D. Communication Initialization Phase

After obtaining a session key, a client can initiate communication with a server. The main objective of this phase is proving ownership of the session key, since the ownership indicates the entity is authenticated and authorized to communicate. Although there are many different ways to prove the ownership, we choose a simple challenge-response handshake, where each entity shows its ability to perform cryptographic operations on randomly generated numbers (nonces) by its counterpart. The random nonces are used to prevent replay

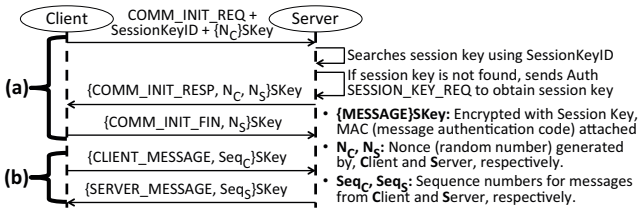

Fig. 7. (a) Communication initialization phase, followed by (b) secure communication phase

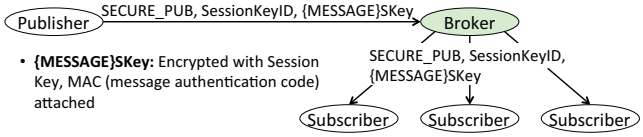

Fig. 8. Alternative secure communication phase for publish-subscribe protocols

attacks. Fig. 7 (a) shows the operations of the communication initialization phase between the client and server.

The communication initialization phase begins with the client's COMM_INIT_REQ with SessionKeyID, a unique identifier for the session key. Note that the client must send the communication initialization request's header and SessionKeyID in clear text, with the client's nonce and its message authentication code (MAC) encrypted by the session key.

Upon receiving COMM_INIT_REQ, the server searches for the session key using SessionKeyID in its cache. The server finds the session key if it is already cached. Otherwise, the server sends SESSION_KEY_REQ to Auth with SessionKeyID specified to obtain the session key. After decrypting COMM_INIT_REQ, the server sends COMM_INIT_RESP with the client's nonce and the server's nonce encrypted with the session key. The client receives COMM_INIT_RESP, decrypts it, and compares the nonce in it against the nonce generated by the client. If the two nonces match, the server is verified to have the session key. In the same way, the client sends COMM_INIT_FIN with server's nonce encrypted, and server verifies the client's ownership of the session key. If either the client or the server is unable to match nonces, communication initialization fails.

E. Secure Communication Phase

After the client and server initialize communication, they can exchange encrypted messages as shown in Fig. 7 (b). This works almost the same as the TLS record layer with application data after TLS handshake. Each message is assigned a sequence number, starting from 0 for the first message, to prevent reply attacks. Every message has MAC attached, encrypted with the symmetric session key.

Our proposed approach also provides an alternative way of secure communication phase for publish-subscribe protocols such as MQTT, as depicted in Fig. 8. The packet for this method of secure communication, SECURE_PUB, includes its header and SessionKeyID in clear text. Any entity with the session key specified by SessionKeyID is authorized to decrypt the encrypted messages. The sender only needs to encrypt and send the message once for all receivers, thus, this scales very well together with one-to-many communication such as broadcasting and publish-subscribe patterns.

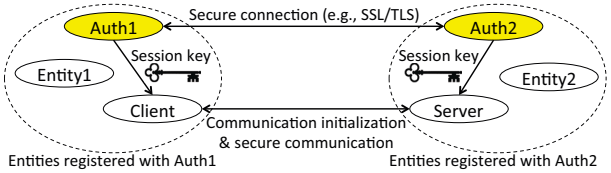

Fig. 9. An example where a client and a server that are registered with two different Auths initialize a secure communication

Lagutin *et al.* [23] introduce other various ways to secure a publish-subscribe network architecture using certificates, including packet level authentication (PLA). Such methods require an entity to either carry large certificates or perform expensive asymmetric key operations for published messages. Compared to these approaches, our approach can significantly reduce the overhead to secure publish-subscribe by using a small and lightweight symmetric session key.

When the published or broadcasted data does not require confidentiality, the TESLA [24] protocol can be used to guarantee data integrity and authenticity for message receivers. The proposed approach can be integrated with TESLA, in a way that Auth establishes an authentication key for a sender and discloses the key to receivers after key disclosure delay.

F. Scaling to Multiple Auths

The proposed approach can also support authentication/authorization between entities that are registered with different Auths. We illustrate this with an example shown in Fig. 9. In this example, a client registered with Auth1 communicates with a server registered with Auth2. Each entity is authorized by its own Auth, that is, the client and the server receive a session key from Auth1 and Auth2, respectively. Auth1 and Auth2 are connected via a secure connection such as SSL/TLS. They work together to distribute the same session key for the client and server.

Fig. 10 describes the detailed authorization process of this example. The client sends SESSION_KEY_REQ to Auth1, specifying the server as a target of communication, as explained in Section IV-C. Auth1 responds to the client's request with a session key, Skey_{CS}, and its unique ID, SKID_{CS}. SKID_{CS} is encoded with information of the session key's generator, in this case, Auth1. Using Skey_{CS} and SKID_{CS}, the client sends COMM_INIT_REQ to the server for initialization of a secure communication, as explained in Section IV-D.

To continue communication initialization, the server requests its own Auth, Auth2, for a session key specified by SKID_{CS}. Auth2 decodes SKID_{CS} and finds that the requested session key was generated Auth1. Auth2 sends AUTH_SESSION_KEY_REQ (a session key request between Auths) to Auth1 specifying the session key ID, SKID_{CS}, via

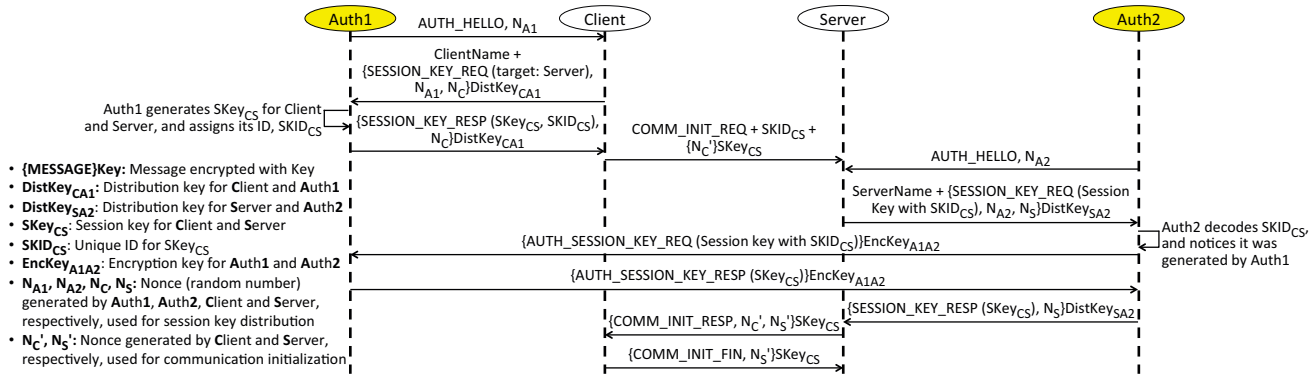

Fig. 10. Details of the example authorization process of a client and a server that are registered with two different Auths (continued from Fig. 9)

a pre-established secure connection such as SSL/TLS. Auth1 replies to Auth2 with AUTH_SESSION_KEY_RESP, which includes $SKey_{CS}$. Auth2 delivers $SKey_{CS}$ to the server, and the server continues to set up a secure connection with the client.

V. Experiments and Results

We compare our approach's effectiveness against TLS with a couple of scenarios that frequently occur in the IoT. We view TLS as an appropriate comparison, because it is widely used and it can support strong cryptography including the public key cryptography for critical components in the IoT. Although we do not conduct experiments with Kerberos, we expect Kerberos will show a similar tendency as TLS, if used with certificates for authentication. Fig. 11 describes the scenarios used for our experiments. In the scenario 1, a client, possibly mobile, is expected to interact with multiple servers one-on-one. In the scenario 2, a publisher, possibly resource-constrained, sends the same messages to multiple subscribers. We assume the entities used for experiments of the proposed approach initially do not have a valid distribution key.

We implement entities for the proposed approach and TLS using Node.js [25], a JavaScript runtime platform. We modify the OpenSSL library included in Node.js to enable logging for cryptography operations. All entities for our experiments run on the local host with different port numbers. We use a packet sniffer, Wireshark (https://www.wireshark.org), to capture network packets generated from our experiments.

To estimate the overall security overhead, we calculate energy consumption caused by security computation and communication. For cryptography operations (RSA-2048, AES-128 and SHA-256), we use the energy cost measured for a mobile device, HP Hx2790, in [26]. To estimate energy

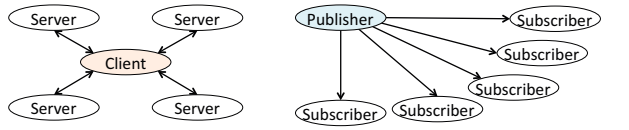

Fig. 11. Experimental scenarios for evaluation

TABLE I
Energy cost model for operations (energy numbers from [26] and [27])

Operation	Energy cost
RSA-2048	91.02 mJ per encrypt/sign operation
	4.41 mJ per decrypt/verify operation
AES-128	0.19 μJ per byte encrypted/decrypted
SHA-256	0.14 μJ per byte digested
Send packet	454 μJ + 1.9 μJ $\times$ packet size (bytes)
Receive packet	356 μJ + 0.5 μJ $\times$ packet size (bytes)

consumption by wireless communication, we use the model and coefficients introduced in [27]. Table I summarizes the energy cost model used for our experiments.

We use the same cryptography algorithms for both approaches to ensure a comparable security guarantee level. For TLS, we choose a cipher suite in TLS version 1.2, which uses RSA for authentication and key exchange, AES-128-CBC for bulk cipher, and SHA-256 for message authentication. Our proposed approach also uses the same algorithms for the same purposes. For asymmetric key pairs, we use X.509 certificates with 2048-bit key and SHA-256 digest.

A. Scenario 1: A client and servers

In experimental scenario 1, a client communicates with multiple servers. We assume these communications should be private to the client and each server. For TLS, the client and each server initiate a TLS connection by exchanging certificates. For the proposed approach, the client requests Auth for session keys to be used for communication with servers. Using a different session key each time, the client initializes secure communication with servers.

For evaluating overhead, we focus on the overhead of the client because the client's overhead increases as the number of servers increases, while the overhead of each server remains the same. Our observation suggests the overhead of secure communication after initialization is almost the same for both TLS and the proposed approach. Hence, we only measure the overhead of communication setup/close.

Table II shows the experimental results for scenario 1 at three different scales, with 16, 32 and 64 servers. We note that difference in the network data sizes between two

TABLE II

SCENARIO 1 RESULTS FOR CLIENT SETUP/CLOSE (PROP.: PROPOSED, ENC: ENCRYPTIONS, DEC: DECRYPTIONS, TX: TRANSMITTED, RX: RECEIVED)

Scale	16 servers		32 servers		64 servers	
Approaches	**TLS**	**Prop.**	**TLS**	**Prop.**	**TLS**	**Prop.**
RSA-2048 (enc/dec)	32/32	2/2	64/64	2/2	128/128	2/2
AES-128 (bytes)	5,120	3,744	10,240	7,392	20,480	14,688
SHA-256 (bytes)	188,976	1,957	377,952	3,349	755,904	6,133
Packets (Tx/Rx)	159/145	135/120	332/300	263/232	650/587	511/449
Tx bytes	56,168	11,031	113,120	21,143	222,502	40,735
Rx bytes	66,808	9,453	134,176	17,805	263,956	34,023

TABLE III

SCENARIO 2 RESULTS FOR PUBLISHER SETUP (BRK.: PROPOSED+BROKER)

Scale	16 subscribers			32 subscribers			64 subscribers		
Approaches	**TLS**	**Prop.**	**Brk.**	**TLS**	**Prop.**	**Brk.**	**TLS**	**Prop.**	**Brk.**
RSA-2048 (enc/dec)	0/49	2/2	2/2	0/97	2/2	2/2	0/193	2/2	2/2
AES-128 (bytes)	22,016	3,424	128	44,032	6,496	128	88,064	12,416	128
SHA-256 (bytes)	179,921	1,556	601	359,153	2,372	601	717,617	3,865	601
Packets (Tx/Rx)	96/96	87/88	11/12	192/192	167/168	11/12	384/384	327/328	11/12
Tx bytes	64,064	6,903	1,184	128,128	12,887	1,184	256,256	24,855	1,184
Rx bytes	51,536	8,557	1,373	103,072	15,981	1,373	206,144	30,829	1,373

TABLE IV

SCENARIO 2 RESULTS FOR PUBLISHING A 256-BYTE MESSAGE

Scale	16 subscribers			32 subscribers			64 subscribers		
Approaches	**TLS**	**Prop.**	**Brk.**	**TLS**	**Prop.**	**Brk.**	**TLS**	**Prop.**	**Brk.**
AES-128 (bytes)	5,120	304	304	10,240	304	304	20,480	304	304
SHA-256 (bytes)	4,832	264	264	9,648	264	264	19,280	264	264
Packets (Tx/Rx)	16/16	16/16	1/1	32/32	32/32	1/1	64/64	64/64	1/1
Tx bytes	6,096	6,064	398	12,192	12,128	398	24,384	24,256	398
Rx bytes	896	896	56	1,792	1,792	56	3,584	3,584	56

approaches can be due to the fact that TLS is a full-fledged protocol with a variety of features, while our approach is still a prototype. Nevertheless, we can see that the number of RSA operations (encryptions/decryptions) for the proposed approach remains constant even with more servers. As seen in Table I, RSA operations cost the most energy. Thus, it indicates our approach scales better than TLS with more private communications with servers.

B. Scenario 2: A publisher and subscribers

A publisher entity publishes encrypted messages to its subscribers in scenario 2. We assume only valid subscribers should be able to decrypt the published messages. Under this assumption, TLS cannot use the message broker of publish-subscribe protocols such as MQTT [5] if the broker is not a subscriber. Therefore, for TLS, we use one-on-one connections between the publisher and subscribers.

For the proposed approach, we can use our secure communication for publishing in Fig. 8, because a MQTT broker cannot see the message without a valid session key. Hence, we carry out experiments in two ways, the proposed approach based on one-on-one communication (marked *Proposed* or *Prop.*), and the proposed approach with the MQTT broker (marked *Proposed+Broker* or *Brk.*). We use an open source MQTT broker, Mosquitto (http://mosquitto.org), for experiments.

As in the scenario 1, we focus on the publisher whose overhead scales with more subscribers. Table III shows the overhead measured for publisher setup at three different scales, with 16, 32 and 64 subscribers. Similar to the scenario 1 results, the proposed approach uses a constant number of RSA operations. When the proposed approach is used with a broker, it uses the same number of packets even when there are more subscribers. We also measure overhead for publishing a 256-byte payload for each approach and each scale, as shown in Table IV. We observe the overhead for the proposed approach with a broker remains constant while that of TLS and the proposed without a broker grows linearly.

C. Energy Cost and Scalability Analysis

With the experimental results and the energy cost model in Table I, we can estimate the overall overhead of each scenario in terms of the energy cost. The estimated overhead in terms of energy consumption is shown in Fig. 12. From the result of scenario 1 for the client setup and close in Fig. 12 (a), we note that the estimated energy consumption of our approach is only 9.62% of TLS for 16 servers. The difference becomes more significant when there are more servers. For 64 servers, the proposed approach is expected to use only 5.09% of energy, compared to TLS. Thus, we estimate that the overhead of the proposed approach grows much more slowly than TLS as the number of servers increases.

The result for scenario 2 for publisher setup in Fig. 12 (b) shows that the setup overhead of the proposed approach scales at a significantly slower rate than TLS, and with the broker, the overhead stays constant. From Fig. 12 (c), we can see that the overhead for publishing a message in scenario 2 grows linearly for both TLS and the proposed approach based on one-one-on communication. However, it is also shown that the proposed approach's overhead for publishing messages can be constant when it works together with the MQTT message broker.

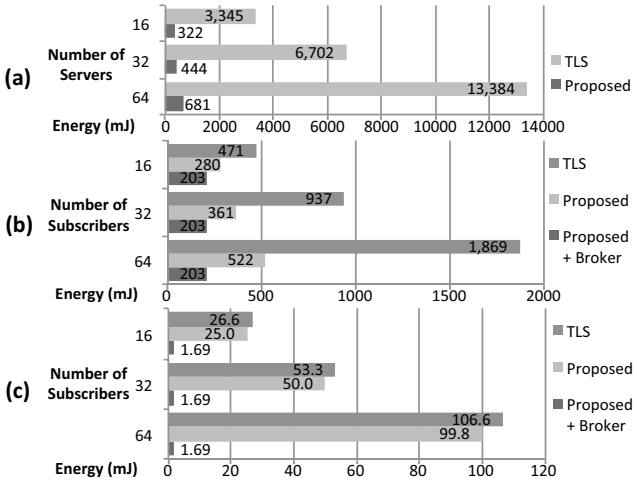

Fig. 12. Estimated energy consumption for (a) scenario 1, client setup/close (b) scenario 2, publisher setup (c) scenario 2, publishing a 256-byte message

TABLE V
HOW THE PROPOSED APPROACH ADDRESSES IOT-RELATED SECURITY REQUIREMENTS INTRODUCED IN SECTION II

IoT Security Requirements	Proposed Approach
Frequent authentication and authorization	Auth controls every secure communication, and it can enforce short key validity periods of session keys
Automated mutual authentication	Auth provides fully automated authentication; no human intervention is required except for entity registration
Intermittent connectivity	Auth allows use of cached session keys
Support for scalability features	Session keys can be shared by more than two entities for one-to-many communication (e.g., publish-subscribe)
Consideration for resource constraints	Small and lightweight symmetric session keys are used for authentication; Auth allows various cryptographic algorithms for resource-constrained devices, including the ones that cannot afford public key cryptography
Privacy	No unique identifier is needed for authentication, thanks to the use of temporary session keys
Dynamic entity registration	An entity can be seamlessly registered/unregistered with Auth, without interrupting other entities

VI. CONCLUSION

In this paper, we propose a secure network architecture to address IoT-related security requirements, as summarized in Table V. The proposed approach supports frequent, automated authentication and authorization by using a local authorization entity called Auth. Auth authorizes registered entities through session key distribution. By caching the session keys and allowing a variety of cryptographic algorithms, even the entities with intermittent connectivity or resource constraints can be authorized effectively. For authentication and authorization, an entity only needs to use temporary session keys provided by Auth. Thus, it does not have to risk exposing its identity by using its unique value such as a certificate, maintaining its privacy. Through experiments, we show our approach has significantly better scalability than SSL/TLS for the scenarios common in the IoT, while providing a comparable level of security as SSL/TLS.

ACKNOWLEDGMENT

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This research was in part supported by an FP7 Marie Curie IOF Action within under the funding ID PIOF-GA-2012-326604 (MODESEC).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012.

[3] P. Traverse, I. Lacaze, and J. Souyris, "Airbus Fly-By-Wire: A total approach to dependability," in *Building the Inform. Soc.*, ser. IFIP Intl. Federation for Inform. Process. Springer, 2004, no. 156, pp. 191–212.

[4] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[5] A. Banks and R. Gupta, "MQTT version 3.1.1," OASIS Standard, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html, Oct 2014.

[6] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," RFC 5246, IETF, Aug. 2008.

[7] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2710–2723, Nov. 2013.

[8] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," RFC 6347, IETF, Jan. 2012.

[9] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," RFC 7252, IETF, Jun. 2014.

[10] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos network authentication service (V5)," RFC 4120, IETF, Jul. 2005.

[11] L. Zhu and B. Tung, "Public key cryptography for initial authentication in Kerberos (PKINIT)," RFC 4556, IETF, Jun. 2006.

[12] W. Du, J. Deng, Y. Han, S. Chen, and P. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *INFOCOM 2004. Twenty-third Annu. Joint Conf. of the IEEE Comput. and Commun. Societies*, vol. 1, Mar. 2004, p. 597.

[13] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili, "A pairwise key predistribution scheme for wireless sensor networks," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 2, pp. 228–258, May 2005.

[14] F. Gandino, B. Montrucchio, and M. Rebaudengo, "Key management for static wireless sensor networks with node adding," *IEEE Trans. on Industrial Informatics*, vol. 10, no. 2, pp. 1133–1143, May 2014.

[15] X. He, M. Niedermeier, and H. de Meer, "Dynamic key management in wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 611–622, Mar. 2013.

[16] X. Zhang, J. He, and Q. Wei, "EDDK: Energy-efficient distributed deterministic key management for wireless sensor networks," *EURASIP Journal on Wirel. Commun. Netw.*, vol. 2011, pp. 12:1–12:11, Jan. 2011.

[17] J.-Y. Huang, I.-E. Liao, and H.-W. Tang, "A forward authentication key management scheme for heterogeneous sensor networks," *EURASIP Journal on Wirel. Commun. Netw.*, vol. 2011, pp. 6:1–6:10, Jan. 2011.

[18] O. K. Sahingoz, "Large scale wireless sensor networks with multi-level dynamic key management scheme," *Journal of Systems Architecture*, vol. 59, no. 9, pp. 801–807, Oct. 2013.

[19] S. H. Erfani, H. H. Javadi, and A. M. Rahmani, "A dynamic key management scheme for dynamic wireless sensor networks," *Security and Communication Networks*, vol. 8, no. 6, pp. 1040–1049, Apr. 2015.

[20] E. Barker, "Recommendation for key management – Part 1: General," *NIST Special Publication 800-57: Part 1 (Revision 4)*, Jan. 2016. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4

[21] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 7–12.

[22] F. Aloul, S. Zahidi, and W. El-Hajj, "Two factor authentication using mobile phones," in *IEEE/ACS Intl. Conf. on Computer Systems and Applications, 2009. AICCSA 2009*, May 2009, pp. 641–644.

[23] D. Lagutin, K. Visala, A. Zahemszky, T. Burbridge, and G. Marias, "Roles and security in a publish/subscribe network architecture," in *2010 IEEE Symp. on Comput. and Commun. (ISCC)*, Jun. 2010, pp. 68–74.

[24] A. Perrig, R. Canetti, J. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *RSA CryptoBytes*, Jul. 2005.

[25] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[26] H. Rifà-Pous and J. Herrera-Joancomartí, "Computational and energy costs of cryptographic algorithms on handheld devices," *Future Internet*, vol. 3, no. 1, pp. 31–48, Feb. 2011.

[27] L. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proc. of IEEE INFOCOM 2001. 20th Annual Joint Conf. of the IEEE Comput. and Commun. Societies.*, vol. 3, 2001, pp. 1548–1557.