Computer Communications 34 (2011) 1243–1257

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A token-bucket based notification traffic control mechanism for IMS presence service

Jianxin Liao [a,*], Jinzhu Wang [a], Tonghong Li [b], Jingyu Wang [a], Xiaomin Zhu [a]

[a] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, P.O. Box 296, Beijing 100876, PR China*
[b] *Department of Computer Science, Technical University of Madrid, Madrid 28660, Spain*

ARTICLE INFO

Article history:
Received 18 August 2010
Received in revised form 26 November 2010
Accepted 31 December 2010
Available online 7 January 2011

Keywords:
Notification traffic control
Delayed update
Presence service
IMS

ABSTRACT

Presence is a service that allows a user to be informed about the specified state of another user. Presence service has become a key enabler for next-generation applications such as instant messaging, push-to-talk and web2.0. However, recent studies show that the notification traffic of presence service causes heavy signaling load on IP multimedia subsystem (IMS) network. This paper introduced a token-bucket based notification traffic control (TNTC) mechanism, which is an application layer solution deployed at the presence server. The TNTC aims at upgrading valid access probability while controlling the notification traffic. A mathematical model of a queuing system is proposed to describe TNTC. We analyzed its main probability features and investigated the effects of different parameters on the performance of TNTC. Extensive simulations verified that TNTC can effectively control notification traffic and perform better than the existing schemes in terms of valid access probability and update arrival rate.

1. Introduction

Presence is a service that allows a user to be informed about the specified state of another user. The specified state, such as online/offline status, disposition (out to lunch, away from the computer), activity status (busy, idle), mood (happy, sad) and location of user, reflects the user's accessibility, availability and will to communicate. Presence has become a key enabler for next-generation services such as push-to-talk (PTT), instant messaging (IM) and web2.0, which have facilitated communications among communities of interest, such as groups of friends, colleagues working on the same projects and families [1,2].

There are four fundamental entities in a presence service [3–5]: a principal, a presentity, a watcher and a presence server, which may exist independently or as part of application servers (e.g., PTT, IM and Web2.0). A principal refers to a user who uses presence service and is the owner of presentities or watchers; a presentity is an entity that is capable of providing state information to presence server; a watcher is an entity that subscribes to or requests the state information about a presentity; and a presence server is a network entity which has three main responsibilities: managing the subscription relationships between watcher and presentity; keeping the latest presentity state; and notifying corresponding watchers when the presentity state is updated.

Fig. 1 illustrates a simplified IP multimedia subsystem (IMS) network architecture for presence service [4]. In this architecture, users (referred to as principals) with several kinds of user equipments (UEs) access presence service. The UE plays the role as a presentity when it provides state information to the presence server. On the other hand, the UE is called a watcher if it accesses other UEs' state information from the presence server. In IMS domain, call session control function (CSCF) is responsible for carrying out the session initiated protocol (SIP) control signaling. It consists of Proxy-CSCF (P-CSCF), which acts as the outbound proxy for UE; Interrogating-CSCF (I-CSCF), which is the entry point in home domain; and Serving-CSCF (S-CSCF), which takes charge of triggering services. As an IMS application server, the presence server mainly interacts with S-CSCF via SIP protocol to collect the updated presentity states and notify them to watchers. Besides that, it also accesses the home subscriber server (HSS) to get the user information through DIAMETER protocol.

Three basic SIP requests surrounding the presence service are presented in Fig. 2 [6,7]: SUBSCRIBE, PUBLISH and NOTIFY. Firstly, the watcher A subscribes to a particular presentity via a SUBSCRIBE request. The presence server internally verifies whether watcher A is authorized to subscribe to this presentity. If so, it acknowledges with a 200 OK response and sends the presentity's current state to watcher A via a NOTIFY request. In the mean while, the presence server generates the subscription relationship between watcher A and the presentity. Secondly, watcher B subscribes to identical

* Corresponding author.
E-mail addresses: liaojianxin@ebupt.com (J. Liao), wangjinzhu@ebupt.com (J. Wang), tonghong@fi.upm.es (T. Li), wangjingyu@ebupt.com (J. Wang), zhuxiaomin@ebupt.com (X. Zhu).

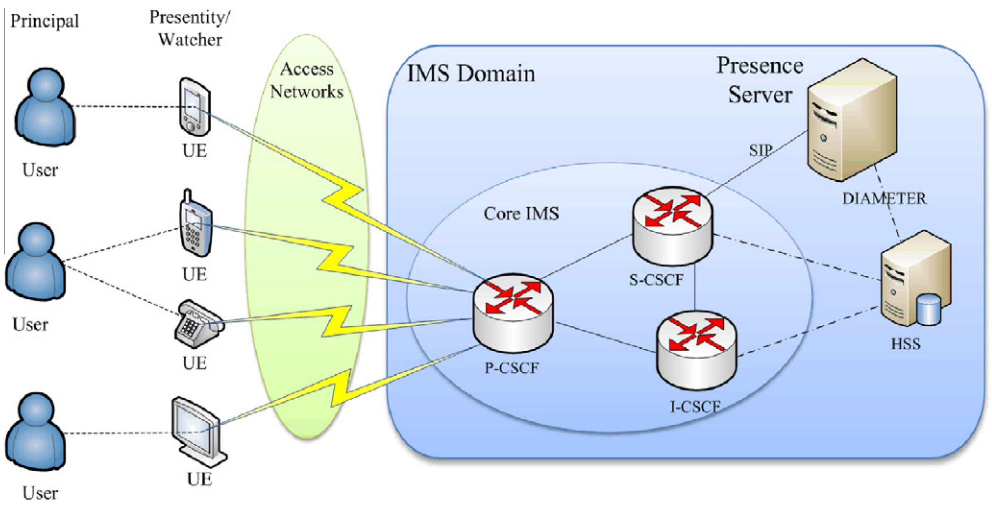

Fig. 1. The IMS architecture for presence server.

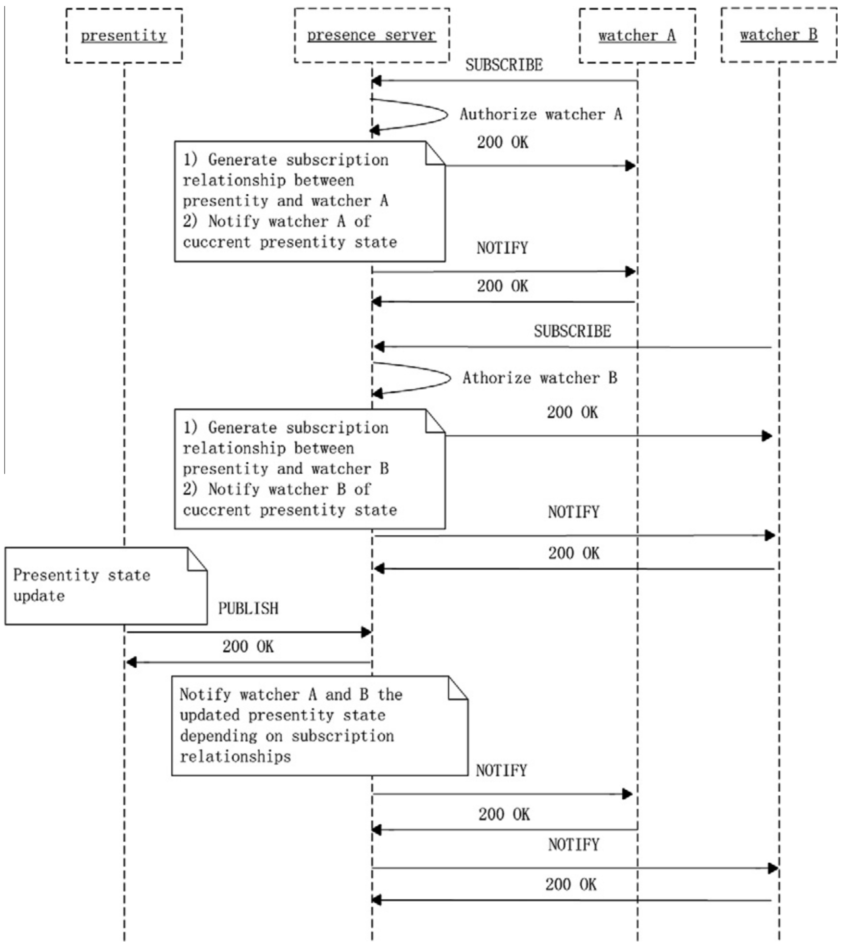
presentity
presence server
watcher A
watcher B
SUBSCRIBE
Authorize watcher A
200 OK
1) Generate subscription relationship between presentity and watcher A
2) Notify watcher A of cuccrent presentity state
NOTIFY
200 OK
SUBSCRIBE
Athorize watcher B
200 OK
1) Generate subscription relationship between presentity and watcher B
2) Notify watcher B of cuccrent presentity state
NOTIFY
200 OK
Presentity state update
PUBLISH
200 OK
Notify watcher A and B the updated presentity state depending on subscription relationships
NOTIFY
200 OK
NOTIFY
200 OK

Fig. 2. Presence call flow.

presentity through the same process as watcher A. Finally, when the presentity's state is updated, it uses a PUBLISH request to notify the presence server. Depending on the subscription relationships, the presence server sends NOTIFY requests to both watcher A and B, notifying them of the renewed presentity state.

A key feature of presence service is always-on [8]. This means that even if the watcher does not actually use the presentity state information after subscribing to the presentity, the presence server still notifies it of the updated presentity state. Considering a presentity often has lots of online watchers in practice, the presence server can generate a high volume of signaling traffic when there are many presentities [9]. A traffic model for presence service is proposed in [1] and it shows that the presence service related load on CSCF could be as high as 50%. Simulation study on the performance of a SIP based presence and instant messaging service for UMTS indicates that high traffic load has bad impact on the IMS network performance and it is necessary to reduce the signaling traffic load [10]. An admission control mechanism is proposed to control the watcher's subscription time such that the traffic load can be reduced [11]. Analysis of the traffic load distribution in the presence service reveals that NOTIFY messages account for the largest portion of the traffic load on a presence server [12,13]. Therefore, controlling notification traffic in IMS presence service is essential for handling the total signal traffic load in IMS core network.

In this paper, we propose and design TNTC, a token-bucket based notification traffic control mechanism. By delaying arrival updates, TNTC can control the notification traffic for presence server. The remainder of this paper is organized as follows: Section 2 surveys related work. Section 3 first proposes the design of TNTC, then analyzes TNTC using queuing theory and calculates its main probability features. Section 4 investigates the effects of different parameters on the performance of TNTC through analytical modeling and uses simulations to compare TNTC with other implementations proposed in [14,15]. Finally, conclusions and possible future work are described in Section 5.

2. Related work

There are several notification traffic control mechanisms proposed in the research literature. Resource list server (RLS) is introduced in [16] to manage all subscriptions to presentities on a resource list. Instead of subscribing to all members on the list and receiving multiple notifications, the watcher subscribes to the resource list on the RLS. The RLS subscribes to the individual presentity. After that, the RLS accumulates the state information of all presentities and notifies watcher through one NOTIFY message. This approach can reduce the number of SUBSCRIBE and NOTIFY messages related to watchers. An extensive RLS mechanism is proposed in [8] to reduce unnecessary notifications for mobile users when they do not need it. When the user deactivates his phone (e.g. locks the phone), the presence server does not send notifications to the user. Once the user activates his phone (e.g. unlocks the phone), his subscription to RLS is renewed. As the RLS never stops collecting presence state information, the user immediately gets the latest presence states. Partial notifications [17] allow a presentity to send only parts of the full state information. As a result, the watcher receives only the changed part of the state information. This does not decrease the number of NOTIFY messages, but significantly reduces the volume of NOTIFY messages. Compressing SIP messages is another technique to reduce the volume of messages sent in the network [18,19]. Usually these compression algorithms substitute words with letters in SIP messages. The compressor builds a dictionary that maps the long expressions to short pointers and sends this dictionary to the decompressor. In this technique, the number of messages transmission is not reduced. An on-demand presence model is proposed in [20], where watchers are not notified every state updating of all the buddies. Instead, watchers only receive state updating of some buddies they are recently interested in. But for other buddies, watchers pull the state information from the presence server at the time they want to call. This on-demand presence model can be useful in reducing notifications at the cost of downgrading the richness of presence information. In [21], a presence network agent (PNA) is implemented to improve the performance of presence service without modifying presence server. It is a new logic entity within the IMS core network that is able to publish the presence state information on behalf of the presentity. The PNA reduces the presence signaling load within the radio access network without downgrading the freshness and richness of the presence information. In inter-domain presence service scenarios [20,22], NOTIFY messages between different domains are aggregated at a specific function entity. As a result, the total number of notifications is reduced.

Besides, research on controlling notification traffic through delaying update is currently in progress. The work in [14] proposed a basic delayed update mechanism to control the notification traffic. In this mechanism, when the presence server receives PUBLISH messages from presentity, the watcher is not notified of the updated information immediately. Instead, the presence server starts a delayed timer with a period *T*. This period is referred to as the delayed threshold. If the presentity state is updated again within the period *T*, the old information in the presence server is replaced by the new one. When the timer expires, the presence server notifies the watcher of the presentity state. Therefore, the notifications for the update messages in *T* are saved and the notification rate is decreased. However, the watcher may access the obsolete information if an access occurs in *T*. This work recommends the delayed threshold to be 5 s. In [15], Chen et al. proposed a mechanism similar to the basic delayed update mechanism, where they set the delayed threshold value to be a random value. This work proposed two specific performance metrics: the probability that the watcher accesses the valid presentity state, and the number of updates saved in the delayed threshold. Both the fixed and exponential delayed thresholds were considered. Simulation results showed that the performance of these two thresholds are almost identical and it is appropriate to select the exponential delayed threshold when the variance of update intervals is small, whereas the fixed delayed threshold should be selected when the variance of update intervals is large. Chi et al. proposed a queuing system with controlled vacation to process NOTIFY messages [13]. That is, NOTIFY messages for updates are put into the queuing system and sent periodically. When there is no NOTIFY message in the queue, the server controlling the queue is on vacation. When the vacation is ended, NOTIFY messages in the queue are sent. Using this mechanism, the presence server can have more time to process other messages (e.g. SUBSCRIBE messages) and the NOTIFY messages are not sent too frequently. Although it does not aim at controlling notification traffic, the vacation time can also play the effect of reducing the notification rate. Niemi et al. proposed a notification rate control (also referred to as event throttling) framework [23], which specifies a mechanism to limit the rate of NOTIFY message received by each watcher. Each watcher can indicate the minimum period of time between two consecutive notifications when subscribing. The presence server maintains a buffer for each watcher. If the arrival interval of updates is less than the minimum period, NOTIFY messages will be delayed in the watcher's buffer. Although this mechanism cannot limit the total NOTIFY message rate sent by presence server when the number of watchers increases, it can reduce the number of NOTIFY messages and limit the NOTIFY message rate received by each watcher.

To sum up, the mechanisms proposed in [8,16–22] reduce the notification traffic volume by optimizing the presence service process, but they cannot limit the notification rate when the number of users or the update rate of presentity state increases. In delayed update mechanisms, [14,15] reduce the number of NOTIFY messages through predefined delayed update time without significant overhead on the presence server. Although they can reduce the notification rate, the predefined delayed update time degrades the valid access probability when updates arrive at a lower rate. The vacation time of [13] is also predefined and it has the same disadvantage as [14,15]. [23] can limit the NOTIFY message receiving rate for each watcher, but the total NOTIFY message sending rate of presence server is impossible to control since each watcher is free to set its rate limitation. Besides, maintaining a buffer and tracing the notification interval for each watcher may result in significant overhead on presence server. Therefore, it is necessary to develop a delayed update mechanism which is able to upgrade valid access probability while control the notification traffic.

3. TNTC design and analytical modeling

3.1. TNTC design

In this section, we propose the design of TNTC. There are three main functional modules in a presence server to process arrival updates as shown in Fig. 3, where TNTC is the module proposed in this paper to control notification traffic and the other two are traditional functional modules.

The presence server records presentity states. Once a presentity state is updated, a PUBLISH message is sent to the presence server. After receiving the message, the presence server puts it into the RESPONSE module, which replies the user with a 200 OK message immediately to indicate that the update has been received successfully. Then, the PUBLISH message is sent into TNTC module, which is responsible for controlling notification traffic. After leaving TNTC, the PUBLISH message goes to the NOTIFICATION module, where the presence server sends each online watcher, who has subscribed to the state of the user, a NOTIFY message to notify them of the updated state.

Fig. 3 shows the model of TNTC consisting of a token bucket and a waiting queue. When TNTC receives a PUBLISH message from the RESPONSE module, it tries to acquire a token from the token bucket for the PUBLISH message. If acquiring a token successfully, TNTC sends the PUBLISH message to the NOTIFICATION module immediately. At the same time, the token acquired is consumed. On the other hand, if the PUBLISH message cannot acquire a token, it is inserted into the waiting queue. Messages in the waiting queue are served in a first-come-first-serve (FIFO) fashion to acquire a token.

Notification traffic control of TNTC consists of two functions: using a token bucket to limit the output rate of PUBLISH messages, and using a waiting queue to reduce the amount of output PUBLISH messages. The token bucket is a common mechanism widely used in computer networks to guarantee QoS, which controls the amount of data injected into the network. Its main functions comprise network traffic shaping and rate limiting [24]. From the knowledge of the token bucket, we know that the maximum PUBLISH message output rate is the token generating rate. Therefore, we can limit the output rate of PUBLISH messages by setting the appropriate token generating rate. If PUBLISH message arrival rate is continuously higher than the token generating rate, the tokens will be consumed. Once there is no token left in token bucket, the arrival PUBLISH messages are put into waiting queue to reduce their output rate. On the other hand, the messages waiting in the queue can reduce the number of output PUBLISH messages. Once a PUBLISH message arrives, it will find in the waiting queue whether there is a message belonging to the same user as itself. If so, it will replace the older message. Thus the notifications for this PUBLISH message are saved and the amount of output PUBLISH messages is decreased.

Compared to the previous notification traffic control methods in [14,15], the fundamental difference between TNTC and the others is the waiting time (also referred to as delayed update time), which in previous methods is predefined while in TNTC is decided by the parameters of TNTC and changes with the varying update arrival rate. We propose an analysis modeling in the following sections to calculate the distribution of waiting time and its expected value in TNTC.

3.2. The parameters of TNTC

The model of TNTC is shown in Fig. 3, and its parameters are described as follows:

- The capacity of token bucket is c. It is the maximum burst volume of notification traffic sent by the presence server. The presence server can process at most c PUBLISH messages at the same time. Therefore, the maximum number of NOTIFY messages sent simultaneously by the presence server is:
$$N_{notify}^{\max} = c \times n_w \tag{1}$$
where n_w is the average number of online watchers for each user.
- The number of online users is N. Thus, the maximum number of waiting messages in TNTC is also N. This is because the arrival PUBLISH message firstly finds in the waiting queue whether there is a message belonging to the same user as itself. If the number of waiting messages is N, there exists a PUBLISH message in waiting queue belonging to the same user as the arrival one. Thus, the arrival PUBLISH message will replace the older one instead of being inserted into the queue as a new waiting message.

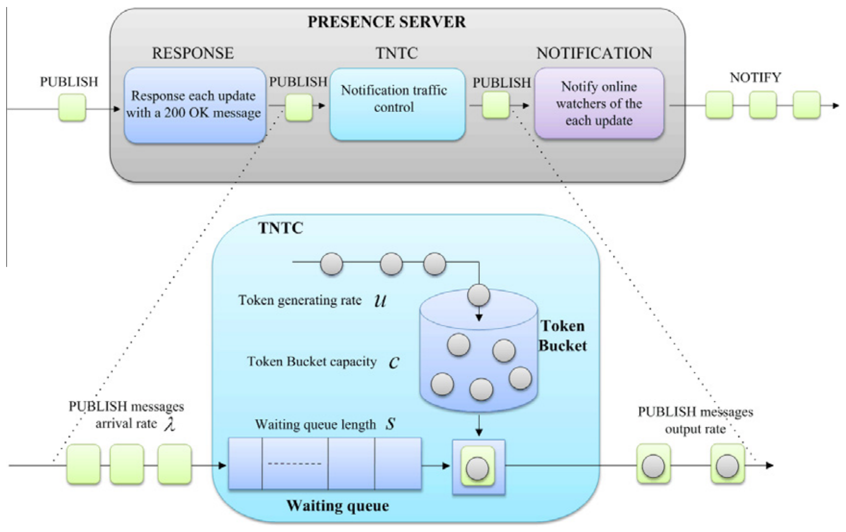
PRESENCE SERVER
RESPONSE
TNTC
NOTIFICATION
PUBLISH
Response each update with a 200 OK message
PUBLISH
Notification traffic control
PUBLISH
Notify online watchers of the each update
NOTIFY
TNTC
Token generating rate u
Token Bucket capacity c
Token Bucket
PUBLISH messages arrival rate λ
Waiting queue length S
PUBLISH messages output rate
Waiting queue

Fig. 3. Update related functional modules in presence server.

- The length of the waiting queue is s. Thus s need not be larger than N as shown above. Different s leads to different waiting time and loss probability of TNTC, which will be discussed in Section 4.1.2.
- From the functional description of RESPONSE module, we can see that the arrival process of PUBLISH messages at presence server is equal to that at TNTC, thus in this paper we do not differentiate these two kinds of arrival processes. Suppose that the arrival process of PUBLISH messages of each user conforms to Poisson distribution with mean λ_a and arrival processes of different users are independent of each other [13]. From the conclusion in [25] that the sum of N independent Poisson variables with the same mean λ_a is a Poisson variable with mean $N \times \lambda_a$, we can conclude that the arrival process of PUBLISH messages of all online users conforms to Poisson distribution with mean λ, as shown in (2):

$$\lambda = N \times \lambda_a \tag{2}$$

- The token generating time interval is set to be exponential distributed with mean u^{-1}. Thus, the token generating process conforms to Poisson distribution with mean u. From the analysis in the last section, we can see that u is the maximum PUBLISH message output rate of TNTC. Therefore, the maximum NOTIFY message sending rate of presence server is:

$$R^{\max}_{notify} = u \times n_w \tag{3}$$

3.3. Analytical modeling

The TNTC can be modeled as a queuing system, where the arrival PUBLISH messages are treated as "customers" and the token bucket is seen as the "service window". Let $S(t)$ denote the number of tokens left in token bucket at time t. Specially, when $S(t)$ is negative, it stands for the number of messages waiting in the queue, e.g. $S(t) = -3$ represents that there are three messages waiting in the queue, and also there is no token left in token bucket. The $S(t)$ is decreased by one unit when a PUBLISH message arrives and increased by one unit at the time a token is generated. We also denote E_k as the state space of $S(t)$, which means $S(t) = k$.

From the definition of $S(t)$, we can conclude that it has Markov property. Since both the inter-arrival times of PUBLISH messages and the token generating time conform to exponential distribution, $S(t)$ is only permitted to transit to neighboring states E_{k+1} or E_{k-1} from state E_k when t tends to be zero. Therefore $\{S(t),\ t \geqslant 0\}$ is the birth–death process as shown in [26], which is the special case of Markov process.

Fig. 4 shows the state-transition-rate diagram of $S(t)$. In order to simplify the expression of the calculation results, we remark this state-transition-rate diagram as shown in Fig. 5, where the state of $S(t)$ is indexed from zero.

In Fig. 5, when $S(t) \leqslant c$, it means there are $c - S(t)$ tokens left in the token bucket and no message is waiting in the queue. All arrival PUBLISH messages are sent to the token bucket. Thus, the rate of updates arriving at token bucket is a fixed value λ. When $S(t) > c$, it means that there is no token left in the token bucket and $S(t) - c$ messages are waiting in the queue. In this circumstance, since the PUBLISH messages arrival process of each user is independent identically distributed, the waiting message in queue belongs to each user with the same probability. Therefore, when a PUBLISH message arrives, the probability of finding message in waiting queue which belongs to the same user as itself is $\frac{S(t)-c}{N}$. As the arrival message will replace the older one if it finds such a message, the probability that the arrival PUBLISH message is inserted into the waiting queue is $\left(1 - \frac{S(t)-c}{N}\right)$. From the theory of decomposition of Poisson process in [25], the rate of updates arriving at token bucket is $\left(1 - \frac{S(t)-c}{N}\right)\lambda$. Since the rate of updates arriving at token bucket is equal to the increasing rate of $S(t)$, and the token generating rate of token bucket is equal to u in each case, which results in constant decreasing rate of $S(t)$, the state-transition-rate diagram of $S(t)$ can be deduced as shown in Fig. 5.

3.3.1. Equilibrium distribution of S(t)

From the above analysis, we can deduce that the rate of updates arriving at token bucket is zero when $S(t)$ is equal to $c + N$, i.e., if the PUBLISH messages of all online users are waiting in the queue, the increasing rate of $S(t)$ is zero. Therefore, from the theory of Markov chain, $S(t)$ has a unique equilibrium distribution and this queue model is stable [26].

Setting

$$p_k^{[s]}(t) = P\{S(t) = k\}, \quad k = 0, 1, \ldots, c + s \tag{4}$$

$$p_k^{[s]} = \lim_{t\to\infty} p_k^{[s]}(t), \quad k = 0, 1, \ldots, c + s \tag{5}$$

where $p_k^{[s]}(t)$ is the probability of $S(t) = k$ when the length of waiting queue is s, and $p_k^{[s]}$ is the limitation of $p_k^{[s]}(t)$ when t goes to infinity. Since $S(t)$ has a unique equilibrium distribution, $p_k^{[s]}$ exists and is unique. From Fig. 5, we obtain the following state-transition-rate equations:

$$p_0^{[s]}\lambda = p_1^{[s]}u \tag{6}$$

$$p_k^{[s]}(\lambda + u) = p_{k-1}^{[s]}\lambda + p_{k+1}^{[s]}u, \quad 1 \leqslant k \leqslant c \tag{7}$$

$$p_k^{[s]}\left[\left(1 - \frac{k-c}{N}\right)\lambda + u\right] = p_{k-1}^{[s]}\left(1 - \frac{k-1-c}{N}\right)\lambda + p_{k+1}^{[s]}u, \quad c < k < c + s \tag{8}$$

$$\sum_{k=0}^{c+s} p_k^{[s]} = 1 \tag{9}$$

where $c > 0$, $s > 0$. The solution of this linear equations array (6)–(8) can be calculated recursively. The result is:

$$p_k^{[s]} = \begin{cases} \rho^k p_0^{[s]}, & 0 \leqslant k \leqslant c \\ \frac{1}{N^{k-c}} \mathrm{P}_N^{k-c} \rho^k p_0^{[s]}, & c < k \leqslant c + s \end{cases} \tag{10}$$

where $\rho = \lambda/u$, $\mathrm{P}_N^{k-c} = \prod_{i=0}^{k-c-1}(N - i)$. From (9) and (10), we can figure out:

$$p_0^{[s]} = \frac{1}{\sum_{k=0}^{c}\rho^k + \sum_{k=c+1}^{c+s}\rho^k \frac{1}{N^{k-c}}\mathrm{P}_N^{k-c}} = \frac{1}{\sum_{j=0}^{c}\rho^j + \sum_{j=1}^{s}\rho^{c+j}\frac{1}{N^j}\mathrm{P}_N^j} \tag{11}$$

Substitute (11) into (10) to obtain the unique equilibrium distribution of $S(t)$:

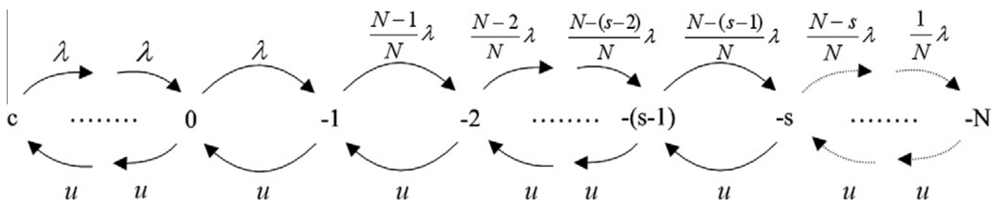

Fig. 4. State-transition-rate diagram indexing from c.

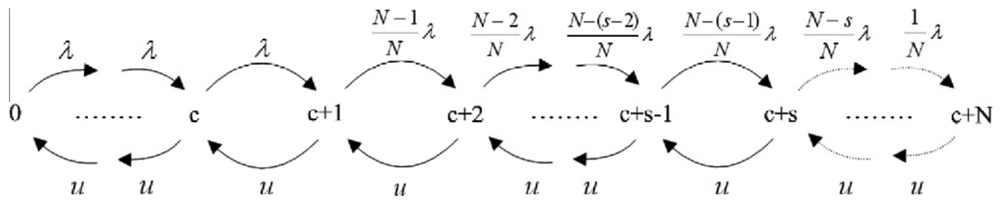

Fig. 5. State-transition-rate diagram indexing from 0.

$$p_k^{[s]} = \begin{cases} \dfrac{\rho^k}{\sum_{j=0}^{c}\rho^j + \sum_{j=1}^{s}\rho^{c+j}\frac{P_N^j}{N^j}}, & 0 \leqslant k \leqslant c \\ \dfrac{\rho^k}{\sum_{j=0}^{c}\rho^j + \sum_{j=1}^{s}\rho^{c+j}\frac{P_N^j}{N^j}} \dfrac{P_N^{k-c}}{N^{k-c}}, & c < k \leqslant c+s \end{cases} \tag{12}$$

3.3.2. Arriving customer's equilibrium distribution of S(t)

As defined in Section 3.3.1, $p_k^{[s]}(t)$ is the probability that $S(t)$ is in state E_k at time t. Stated differently, $p_k^{[s]}(t)$ is the probability that an outside observer, who observes $S(t)$ at time t, will find that $S(t)$ is in state E_k. Now consider the state distribution as seen by the arriving customers (e.g., arriving PUBLISH messages), let $\pi_k^{[s]}(t)$ be the probability that $S(t)$ is in state E_k just prior to time t, where t is now an arrival epoch. $\left\{\pi_k^{[s]}(t)\right\}$ is thus the distribution that represents the viewpoint of the arriving customer at time t. We define $\pi_k^{[s]}$ as:

$$\pi_k^{[s]} = \lim_{t\to\infty} \pi_k^{[s]}(t), \quad k = 0, 1, \ldots, c+s \tag{13}$$

Referring to [25], we can draw a conclusion, which is deduced in Appendix A, as follows:

$$\pi_k^{[s]} = \frac{\lambda_k p_k^{[s]}}{\sum_{j=0}^{c+s} \lambda_j p_j^{[s]}}, \quad k = 0, 1, \ldots c+s \tag{14}$$

where λ_k denotes the rate of updates arriving at token bucket when $S(t)$ is in state E_k, $\left\{\pi_k^{[s]}\right\}$ denotes arriving customer's equilibrium distribution and $\left\{p_k^{[s]}\right\}$ denotes outside observer's equilibrium distribution [25]. From the Fig. 5, we can write:

$$\lambda_k = \begin{cases} \lambda, & 0 \leqslant k \leqslant c \\ \left(1 - \frac{k-c}{N}\right)\lambda, & c < k \leqslant c+s \end{cases} \tag{15}$$

Furthermore, we obtain the average PUBLISH messages arrival rate $\bar{\lambda}$:

$$\bar{\lambda} = \sum_{j=0}^{c+s} \lambda_j p_j^{[s]} \tag{16}$$

When the length of waiting queue is s, an arrival PUBLISH message is dropped by TNTC if and only if the arrival one finds that there are s messages waiting in the queue and there is no message in queue belonging to the same user as itself. Thus, the message loss rate $P_{loss}^{[s]}$ is:

$$P_{loss}^{[s]} = \pi_{c+s}^{[s]} \tag{17}$$

Substituting (14) and (16) into (17), we have:

$$P_{loss}^{[s]} = \frac{\lambda_{c+s} p_{c+s}^{[s]}}{\bar{\lambda}} \tag{18}$$

As a special case, when $s = N$, we can see $\lambda_{c+N} = 0$ from Fig. 5. Thus the rate of updates arriving at token bucket is zero, when $S(t)$ is in state E_{c+N}. As shown in (18), $\lambda_{c+N} = 0$ then leads to $P_{loss}^{[N]} = 0$. Therefore, we can conclude that if we set $s \geqslant N$, the TNTC is lossless.

3.3.3. The number of waiting PUBLISH messages

Let L_w denote the number of PUBLISH messages waiting in the queue. In our system, when $S(t) \leqslant c$, $L_w = 0$ because there are tokens left in the token bucket and no message waiting in the queue. When $S(t) > c$, $L_w = S(t) - c$ because there is no token left in the token bucket and the number of messages waiting in the queue is $S(t) - c$. Therefore, the distribution of L_w is:

$$P\{L_w = k\} = \begin{cases} p_{c+k}^{[s]}, & 0 < k \leqslant s \\ \sum_{j=0}^{c} p_j^{[s]}, & k = 0 \end{cases} \tag{19}$$

The mean $\overline{L_w}$ and standard deviation $\delta(L_w)$ of L_w are:

$$\overline{L_w} = \mathrm{E}(L_w) = \sum_{k=1}^{s} k p_{c+k}^{[s]} \tag{20}$$

$$\mathrm{E}\left(L_w^2\right) = \sum_{k=1}^{s} k^2 p_{c+k}^{[s]} \tag{21}$$

$$\delta(L_w) = \sqrt{\mathrm{E}\left(L_w^2\right) - \mathrm{E}^2(L_w)} \tag{22}$$

3.3.4. The waiting time of arrival PUBLISH message

Let T_w denote the waiting time of arrival PUBLISH message. From the analysis of TNTC in Section 3.1, we find that T_w is only subject to the token generating time and the number of waiting messages in the queue. Thus we can conclude that if there is no token left in token bucket when a PUBLISH message arrives at TNTC, the waiting time of this message is the time that token bucket needs to generate a certain amount of tokens, where the number of these tokens is equal to the number of messages waiting in front of this message plus one; if there are tokens in the token bucket when a PUBLISH message arrives, its waiting time is zero. For example, if an arrival PUBLISH message finds that there are three messages waiting in the queue, the waiting time of this message is the time token bucket needs to generate four tokens; if an arrival PUBLISH message finds that there is no message waiting in the queue and no token in token bucket, the waiting time of this message is the time token bucket needs to generate one token; if an arrival PUBLISH message finds that there are tokens in token bucket, the waiting time is zero.

Let Q be the number of messages waiting for a token when a PUBLISH message arrives. Then, from the theorem of total probability [25], we have:

$$P\{T_w > t\} = \sum_{i=0}^{s-1} P\{T_w > t | Q = i\} P\{Q = i\} \tag{23}$$

If $Q = s$, the arrival PUBLISH message will either replace the older one in waiting queue belonging to the same user as itself or be dropped. Thus, For each arrival PUBLISH message, if it is permitted to wait in the queue, the maximum number of waiting messages in viewpoint of this arrival one is $s - 1$. That is the reason why the maximum value of Q in (23) is $s - 1$.

From the definition of Q, we have:

$$P\{Q = i\} = \pi_{c+i}^{[s]} \tag{24}$$

On the condition that the number of waiting messages in viewpoint of arrival PUBLISH message is i, the waiting time distribution of this arrival PUBLISH message is shown as (25), which is deduced in Appendix B.

$$P\{T_w > t|Q = i\} = \sum_{j=0}^{i} \frac{(ut)^j}{j!} e^{-ut} \tag{25}$$

Substituting (24) and (25) into (23), we can conclude that the distribution of waiting time is:

$$P\{T_w > t\} = \sum_{i=0}^{s-1} \left(\pi_{c+i}^{[s]} \sum_{j=0}^{i} \frac{(ut)^j}{j!} e^{-ut} \right) \tag{26}$$

Now calculate the mean and standard deviation of waiting time. Because the expression of waiting time distribution is complex, it is difficult to directly calculate its mean and standard deviation. Fortunately, the theorem of Laplace transform can help us to solve this problem. The Laplace transform of probability density function of waiting time is shown in (27), which is deduced in Appendix C.

$$f^*(s') = -s' \sum_{i=0}^{s-1} \left(\pi_{c+i}^{[s]} \sum_{j=0}^{i} \frac{u^j}{(s'+u)^{j+1}} \right) \tag{27}$$

where $f^*(s')$ is the Laplace transform of the probability density function of T_w and s' is the argument of Laplace transform. From [27], we can see that the appropriate derivative of the Laplace transform evaluated at its argument $s' = 0$ gives rise to moments of the random variable, as shown:

$$f^{*(k)}(0) = \left. \frac{d^k f^*(s')}{ds'^k} \right|_{s'=0} = (-1)^k \mathrm{E}\left[T_w^k\right] \tag{28}$$

Thus, we have the mean $\overline{T_w}$ and standard deviation $\delta(T_w)$ of T_w:

$$\overline{T_w} = \mathrm{E}[T_w] = (-1) \left. \frac{df^*(s')}{ds'} \right|_{s'=0} = \sum_{i=0}^{s-1} \pi_{c+i}^{[s]} \frac{i+1}{u} \tag{29}$$

$$\mathrm{E}[T_w^2] = \left. \frac{d^2 f^*(s')}{ds'^2} \right|_{s'=0} = \sum_{i=0}^{s-1} \pi_{c+i}^{[s]} \frac{(i+1)(i+2)}{u^2} \tag{30}$$

$$\delta(T_w) = \sqrt{\mathrm{E}\left[T_w^2\right] - \mathrm{E}^2[T_w]} \tag{31}$$

Note that there is an easier method of calculating the mean of waiting time. From the Little's result in [26]: "the average number of customers in a queuing system is equal to the average arrival rate of customers to that system, times the average time spent in that system", using (16) and (20), we have:

$$\overline{T_w} = \frac{\overline{L_w}}{\bar{\lambda}} = \frac{\sum_{k=1}^{s} k p_{c+k}^{[s]}}{\sum_{j=0}^{c+s} \lambda_j p_j^{[s]}} = \sum_{i=0}^{s-1} \pi_{c+i}^{[s]} \frac{i+1}{u} \tag{32}$$

The result of (32) is the same as (29).

In TNTC, the maximum waiting time occurs when an update arrives at the system and finds that there are $s-1$ messages waiting in the queue. Thus when $\pi_{c+s-1}^{[s]} = 1$ and $\pi_{c+i}^{[s]} = 0$ $(i = 0, 1, \ldots, s-2)$, the average waiting time is maximal and it is calculated using (29):

$$\mathrm{E}[T_w]_{\max} = \pi_{c+s-1}^{[s]} \frac{s-1+1}{u} = \frac{s}{u} \tag{33}$$

4. Performance evaluation

In this section, firstly, we investigate the performance of TNTC with different parameters, based on the analysis in Section 3. Secondly, we use the simulation experiments to validate the analytical modeling of Section 3.3 and compare TNTC with other notification traffic control methods proposed in [14,15].

4.1. Mathematic analysis

In this section, depending on the analytical modeling of Section 3.3, we investigate the effects of different parameters defined in Section 3.2 on the performance of TNTC by using Matlab. The results of analysis give a better understanding of performance of TNTC.

4.1.1. Effects on average waiting time

When investigating the effects of different parameters on average waiting time, we set $s = N$, which means that the length of waiting queue is equal to the number of online users and TNTC is lossless as shown in Section 3.3.2.

Fig. 6 plots the effects of λ_a and c on E[T_w]. Clearly, as λ_a increases, E[T_w] increases. We explain this phenomenon as follows. The increase of λ_a results in more messages waiting in the queue, which leads to the increase of E[T_w]. When λ_a is small, $L_w < s$. In this circumstance, as λ_a increases, E[T_w] increases significantly with the remarkable increasing of L_w. On the other hand, when large λ_a is observed, L_w is close to s and will not increase significantly. In this circumstance, the arrival PUBLISH message will probably replace the older one in the waiting queue. Thus as λ_a increases, the increasing rate of E[T_w] decreases and E[T_w] gets closer to the maximum average waiting time E[T_w]$_{\max}$. This phenomenon reflects an important feature of TNTC: the average waiting time will not continuously increase as the state update rate of each user increases. Considering the conclusion demonstrated in [15] that longer average waiting time degrades the valid access probability, which is an important indicator of the user experience, we can conclude that TNTC ensures the user experience even if the state update rate of each user continuously increases.

Fig. 7 shows the effects of N and c on E[T_w]. We can see that E[T_w] continuously increases as N increases. It is explained as follows. The increasing N also results in more messages waiting in the queue, which leads to the increase of E[T_w]. We set $s = N$ to guarantee that TNTC is lossless, s thus increases as N increases. Therefore, when N increases, L_w keeps increasing as the increase of s, which leads to the continuous increase of E[T_w]. In order to guarantee that the valid access probability is higher than some threshold, we calculate the proper value of average waiting time as shown in [15]. Depending on that value, the system capacity, which is the maximum number of online users the presence server can support, is determined by using the relationship as shown in Fig. 7.

From Figs. 6 and 7, we can also see the effects of c on E[T_w]: when $\lambda \leqslant u$, E[T_w] increases as c decreases; when $\lambda > u$, E[T_w] is not sensitive to the value of c. The reason is as follows: if $\lambda \leqslant u$, there are tokens left in token bucket when system is stable. Since the arrival PUBLISH message will be sent without delay when a token is acquired, the more tokens left in token bucket, the shorter waiting times the arrival messages have. Thus E[T_w] increases as c decreases. On the other hand, if $\lambda > u$, there is no token left in token bucket when system is stable. Therefore, c has little effect on E[T_w].

4.1.2. Effects of waiting queue length

Let ρ denote the utilization factor, which is the ratio between λ and u as mentioned in (10). Fig. 8 shows the distribution of L_w with different ρ. In this figure, we set $c = 10$, $N = 20$, $s = 20$. It can be seen that when $\rho \leqslant 1$, L_w tends to be zero, and L_w increases as ρ increases. This phenomenon is explained as follows. When $\rho \leqslant 1$ (i.e., $\lambda \leqslant u$), it is more likely that there are tokens left in token bucket. At this moment, the arrival PUBLISH message will be sent to the

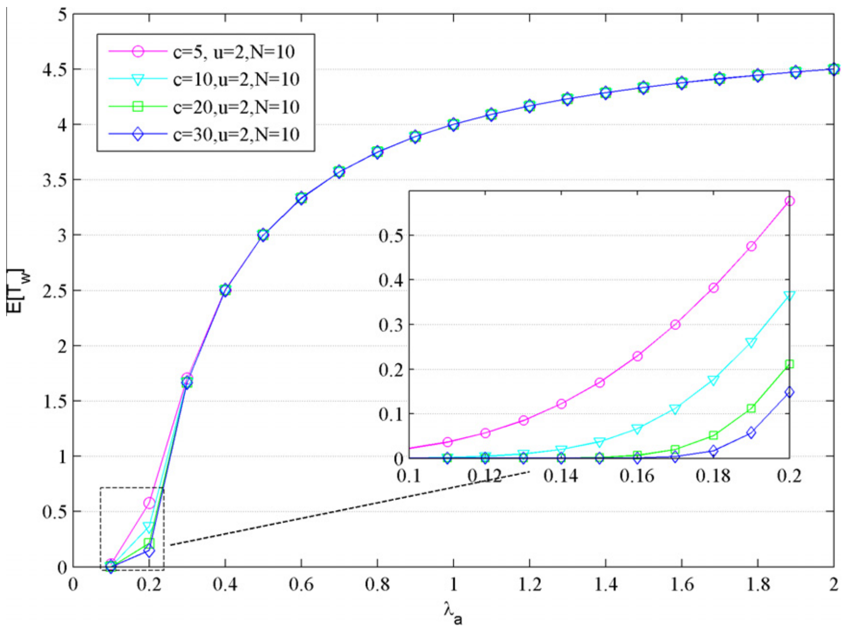

Fig. 6. Effects of λ_a and c on E[T_w].

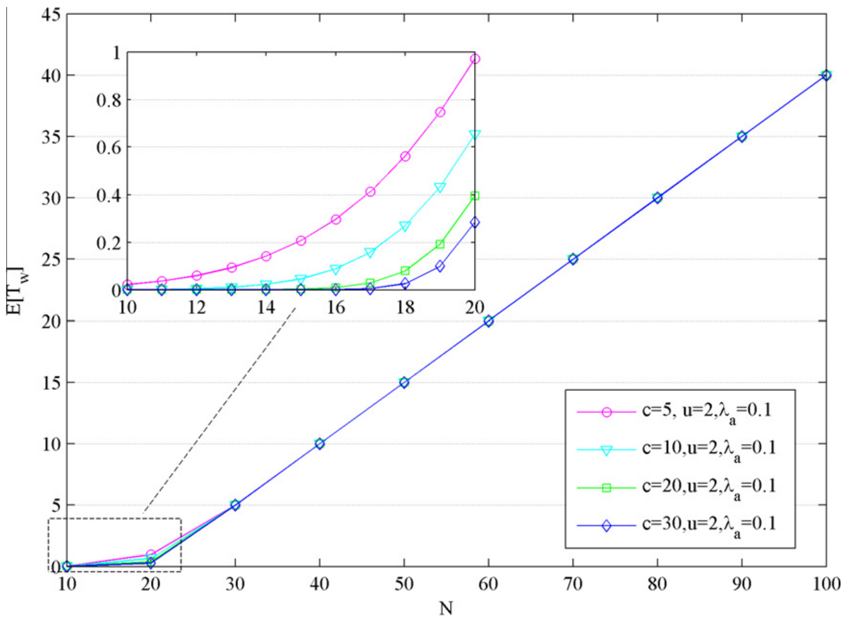

Fig. 7. Effects of N and c on E[T_w].

NOTIFICATION module without delay, and thus L_w tends to be zero. When $\rho > 1$ (i.e., $\lambda > u$), it is more likely that there is no token left in token bucket and there are messages waiting in the queue, thus L_w is greater than zero. When ρ continuously increases, the difference between λ and u increases, which then leads to the increase of L_w.

It is worthwhile to note that although setting $s = N$ can guarantee TNTC is lossless, in some cases s can be set to be smaller than N in order to achieve this goal. For example, when $\rho = 2$, the probability, that L_w is equal to 18, 19 or 20, tends to be zero. Therefore, $s = 17$ is enough for us to avoid the waste of storage resource while control the loss probability under certain threshold.

It is important to select the appropriate value for parameter s in TNTC. Fig. 9 shows the effects of different s values on the message loss probability and the probability that the message waiting time is greater than a given value (e.g. 5 s). We set $\lambda_a = 0.2$, $u = 2$, $N = 20$, $c = 10$. From the figure, it can be seen that as s increases, the message loss probability decreases and the probability that the waiting time is greater than 5 s increases. Therefore, among all the possible values of s meeting the requirements of both message loss and waiting time, the minimum one is selected as the value of s in practice.

4.2. Simulation results

The proposed model of TNTC has been implemented in the discrete-event simulator SIMPROCESS [28]. The purpose of this simulation is twofold: firstly, to validate the accuracy of analytical modeling in Section 3.3; secondly, to compare TNTC with other

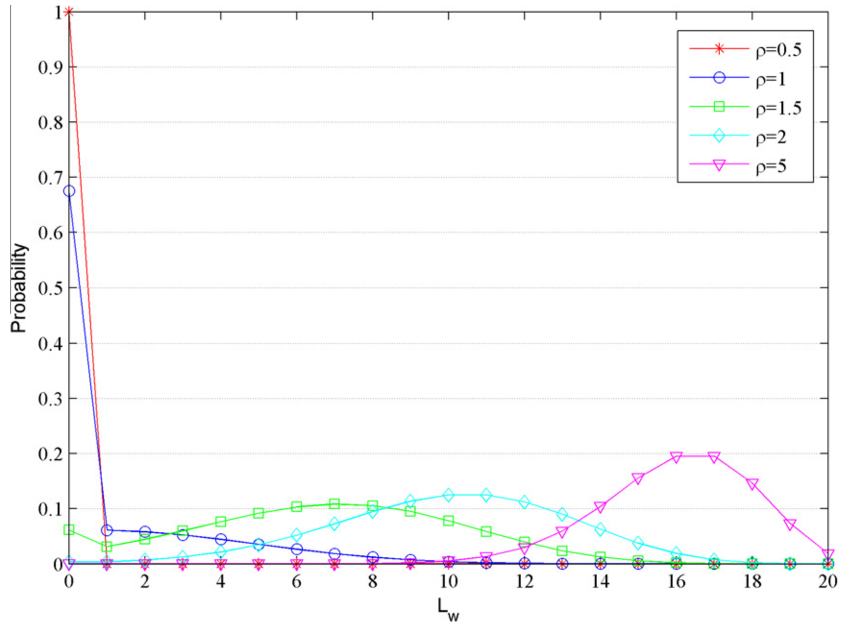

Fig. 8. Distribution of L_w with different ρ.

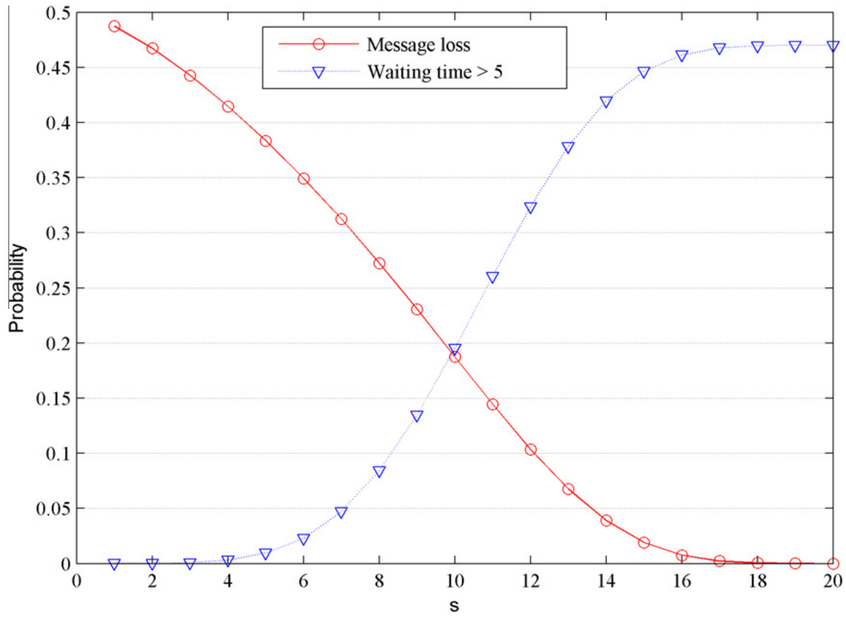

Fig. 9. Effects of s on message loss probability and waiting time.

notification traffic control methods. In our simulation, we created a network consisting of one presence server and N online users. Both the state update interval of each user and token generating interval conform to the exponential distribution, and the PUBLISH messages in the waiting queue are served in a FIFO fashion.

4.2.1. Analytical model validation

In this section, we set $u = 3$, $s = 10$, $N = 10$, $c = 10$.

The calculated results and simulated results under different λ_a in terms of the mean and standard deviation of L_w are compared in Fig. 10. We set the $\alpha(S) = \frac{|S'-S|}{S}$ as the difference between the simulated result S' and calculated result S. It can be seen that $\max \alpha(E[L_w]) \leqslant 0.067$ and $\max \alpha(\delta[L_w]) \leqslant 0.043$, which indicates that (19) can accurately describe the distribution of L_w.

Similar comparisons of T_w are shown in Fig. 11. We can see that $\max \alpha(E[T_w]) \leqslant 0.093$ and $\max \alpha(\delta[T_w]) \leqslant 0.027$, which indicates that (26) is a pretty accurate description of the distribution of T_w.

The distribution of L_w and that of T_w are two key indicators for a queuing system. From the above experiments, we find that the calculated results of these two indicators are consistent with the simulation results. Thus, we can conclude that the analytical modeling proposed in Section 3.3 is an accurate description of TNTC.

4.2.2. Compare with other methods

In this section, we compare TNTC with other notification traffic control methods. In [14], the fixed delay for each arrival PUBLISH message is suggested by IETF, we denote it as the Fixed method. In [15], the delay for each arrival PUBLISH message is proposed

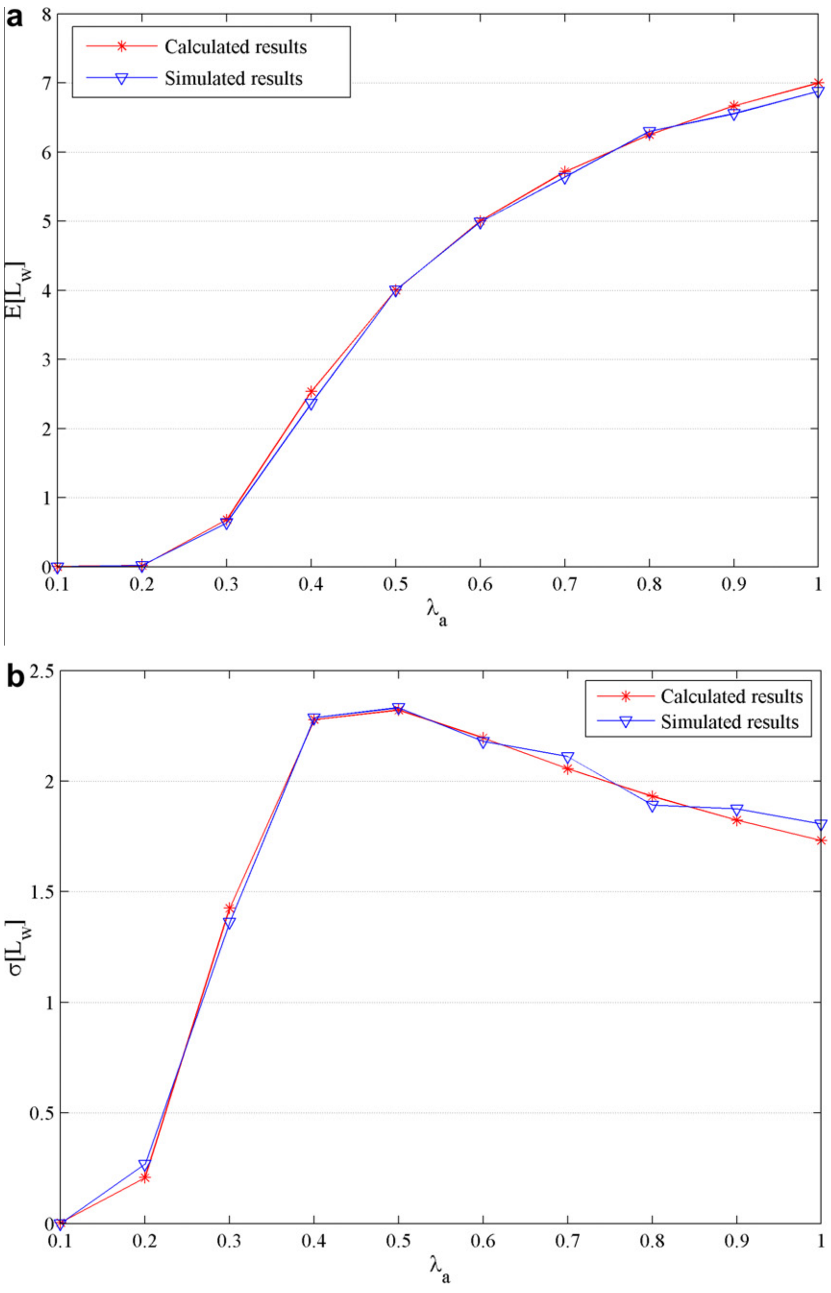

Fig. 10. Comparison between calculated results and simulated results in terms of E[L_w] and $\delta[L_w]$. (a). The mean of L_w, (b). The standard deviation of L_w.

to conform to exponential distribution, which is denoted as the Exp method. Both of these methods are named as timer methods and we adopt E[T_w] as shown in Section 3.3.4 to denote the delayed threshold.

In order to make a comprehensive performance comparison, we adopt an important evaluation metric proposed in [15]: valid access probability p, which stands for the probability that each watcher accesses the valid presence state information. It is calculated as follows:

$$p = \frac{1}{1 + \lambda_a \times \mathrm{E}[T_w]} \tag{34}$$

Besides, we define a new metric: output rate *OR*, which stands for the PUBLISH message output rate of TNTC and timer methods, to indicate the notification traffic load that presence server may generate on IMS network. From Section 3.1, we see that *OR* is equal to the rate of updates arriving at the NOTIFICATION module. Therefore, we can calculate the NOTIFY message sending rate of presence server as:

$$R_{notify} = OR \times n_w \tag{35}$$

OR is an important metric because if *OR* is large, the high notification traffic sent by the presence server may cause the congestion in IMS network, which will then degrade the metric p since the NOTIFY messages sent by the presence server may not be received by users in time due to network congestion. Furthermore, network congestion also has bad impact on other IMS services. Thus, an appropriate notification traffic control method must be able to control *OR* under certain threshold. The maximal threshold of *OR* is referred to as $OR_{\max}$.

For a presence server, λ_a and N are the parameters that influence the total update arrival rate λ. Thus, we design experiments to

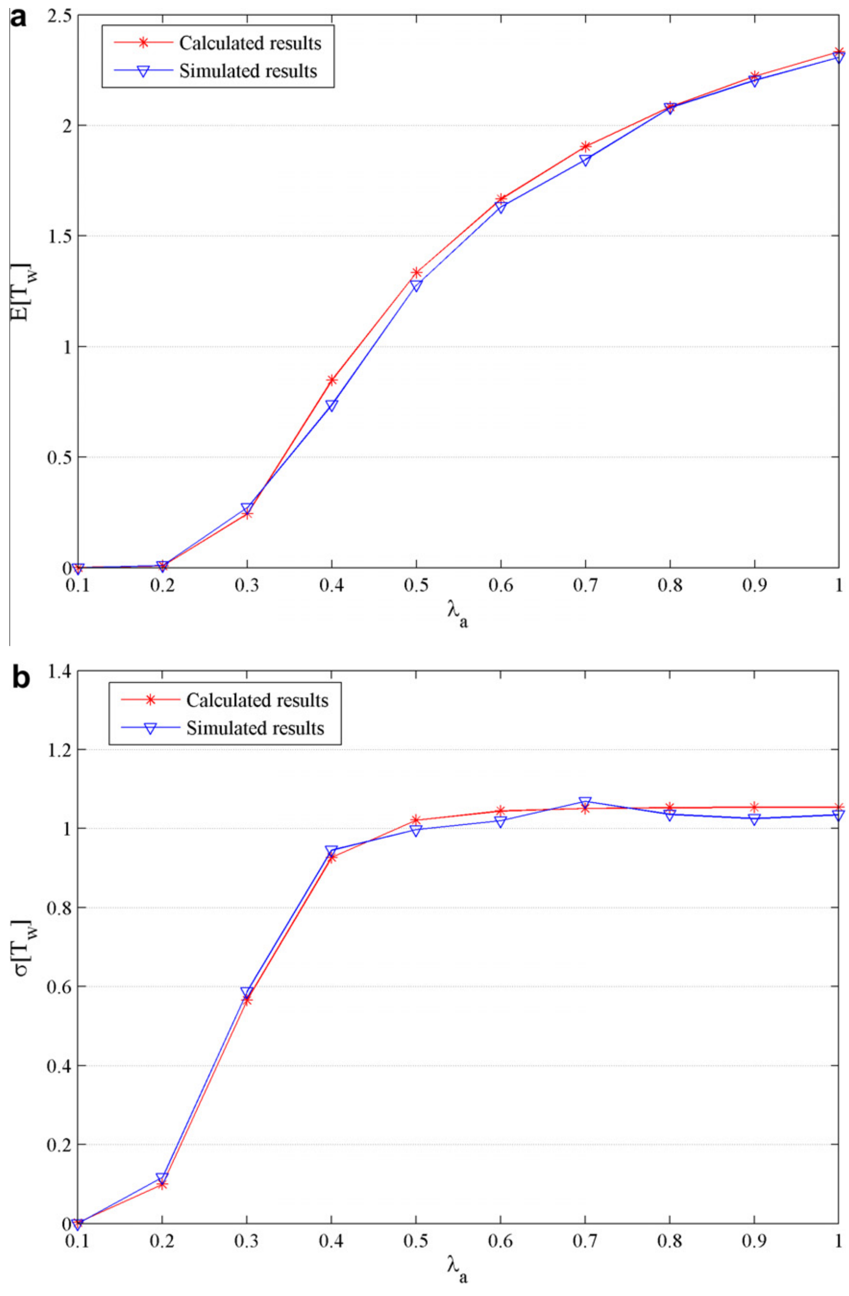

Fig. 11. Comparison between calculated results and simulated results in terms of E[T_w] and δ[T_w]. (a). The mean of T_w, (b). The standard deviation of T_w.

compare TNTC with timer methods by varying the values of these two parameters.

In the first experiment, we set $OR_{\max} = 2$, $N = 10$ and λ_a varies between 0.1 and 2.

In timer methods, the minimum delayed threshold $\mathrm{E}[T_w]_{\min}$ is calculated as follows:

$$\mathrm{E}[T_w]_{\min} = \frac{1}{OR^a_{\max}} = \frac{1}{\frac{OR_{\max}}{N}} = \frac{N}{OR_{\max}} \tag{36}$$

where $OR^a_{\max}$ denotes the maximum PUBLISH message output rate for each user. Referring to (34), we can see that p increases as $\mathrm{E}[T_w]$ decreases. In order to obtain higher p while ensuring that OR is under $OR_{\max}$, we set $\mathrm{E}[T_w]$ as $\mathrm{E}[T_w]_{\min}$ for timer methods, which is equal to 5 s.

In TNTC, we keep $s = N$ to ensure that the TNTC is lossless and set $u = OR_{\max}$ to limit OR. We observe that $OR \leqslant u = OR_{\max}$ when $c = 0$. In the experiment, we set $c = 0$ and explain its reason as follows. When $\lambda \leqslant u$, we can see from Section 4.1.1 that the larger c, the smaller $\mathrm{E}[T_w]$ in TNTC, which further results in higher p and OR. As $OR \leqslant \lambda$ and $\lambda \leqslant u = OR_{\max}$, OR is always smaller than $OR_{\max}$. On the other hand, when $\lambda > u$, from the analysis in Section 4.1.1, we conclude that the value of c has little effect on $\mathrm{E}[T_w]$, which means that p and OR are not sensitive to the value of c. To sum up, if TNTC with $c = 0$ has higher p and OR than timer methods while its OR is under $OR_{\max}$, we have a good reason to believe that TNTC with $c > 0$ also has higher p and OR than timer methods while its OR is under $OR_{\max}$.

From (33), we can calculate that the maximum average waiting time in TNTC is 5 s, which is equal to the delayed threshold in timer methods. Thus the average waiting time in TNTC is smaller than that in timer methods in the first experiment.

In TNTC p can be calculated as follows: first using (29) to obtain $\mathrm{E}[T_w]$, and then using (34) to obtain p. We compare the calculated

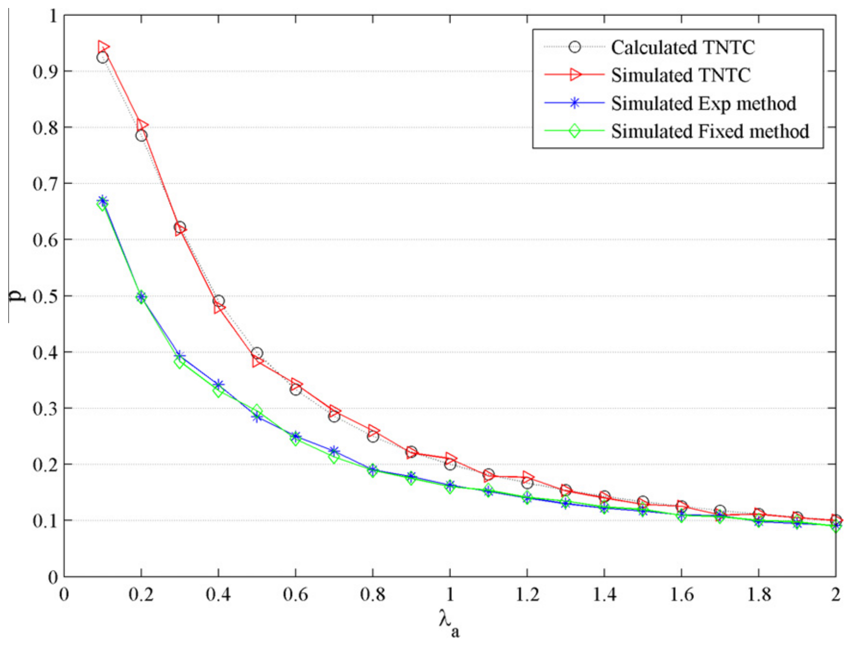

Fig. 12. Comparison of p with varying λ_a between TNTC and timer methods.

results and simulated ones in terms of p in Fig. 12, which shows that the calculated results match quite well with the simulated ones. Thus, we conclude that this approach of calculating p is accurate in TNTC.

From Figs. 12 and 13, we can see that with varying λ_a, p in TNTC is higher than that in timer methods while OR in all methods is under $OR_{\max}$. This is because the E[T_w] in TNTC is smaller than that in timer methods, which further leads to higher p in TNTC. Besides, in Fig. 13, the OR in TNTC is higher than that in timer methods, which indicates that the TNTC makes better use of available bandwidth. On the other hand, we can also see in Fig. 12 that if the p in TNTC and timer methods are kept the same, the update arrival rate of each user which presence server can handle in TNTC is higher than that in timer methods. Thus, TNTC can deal with higher update arrival rate for user.

In the second experiment, we set $OR_{\max} = 2$, $\lambda_a = 0.1$ and N varies between 1 and 30. In this experiment, the maximum number of online users the system can support, which is denoted as $N_{\max}$, is equal to 30.

The PUBLISH message output rate for each user OR^a is calculated as:

$$OR^a = \lambda_a \times p \tag{37}$$

Besides, OR can be calculated as:

$$OR = OR^a \times N \tag{38}$$

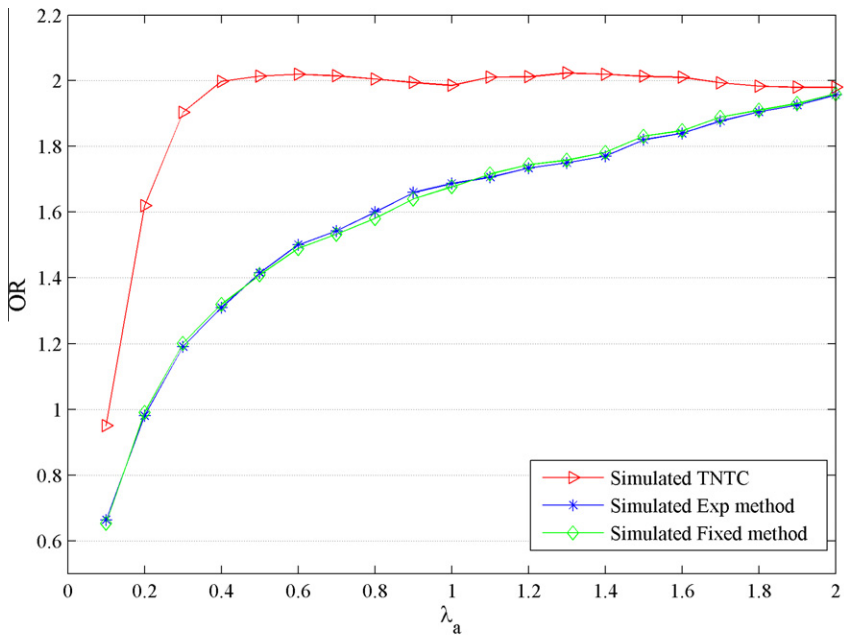

Fig. 13. Comparison of OR with varying λ_a between TNTC and timer methods.

In timer methods, *OR* increases as *N* increases. In order to keep *OR* under threshold OR_{max}, we make sure $OR \leqslant OR_{max}$ when $N = N_{max} = 30$. Considering that *p* increases as *OR* increases, in order to obtain higher *p*, we set *OR* to be the maximum value OR_{max} when $N = N_{max} = 30$. Thus, *p* is calculated by using (37) and (38). Further referring to (34), we can obtain $E[T_w]$, which is equal to 5 s. This delay value is used for timer methods in this experiment for all values of *N*.

In TNTC, we set $s = N_{max}$, $u = OR_{max}$ and $c = 0$. The reason is similar to that in the first experiment.

From Figs. 14 and 15, we observe that with varying *N*, the calculated results match quite well with the simulated results and *p* in TNTC is higher than that in timer methods while *OR* in all methods is under OR_{max}. Besides in Fig. 15, *OR* in TNTC is also higher than that in timer methods, and thus TNTC makes better use of available bandwidth.

Note that the behavior of TNTC and timer methods converge to the same results in terms of *p* and *OR* when *N* = 30. This can be explained as follows: in timer methods, $E[T_w]$ is set to be 5 s for all values of *N*. While in TNTC due to the characteristic of token bucket, in order to guarantee that *OR* is under threshold OR_{max}, we only need to set *u* to be OR_{max}. Referring to (29), when *N* = 30, the $E[T_w]$ in TNTC is calculated to be 5 s. Thus, when *N* = 30, first using (34), we can see that TNTC has the same *p* as timer methods, then referring to (37) and (38), we conclude that both TNTC and timer methods have same *OR*, which is OR_{max}.

From above two experiments, we conclude that while keeping PUBLISH message output rate under threshold OR_{max}, TNTC can

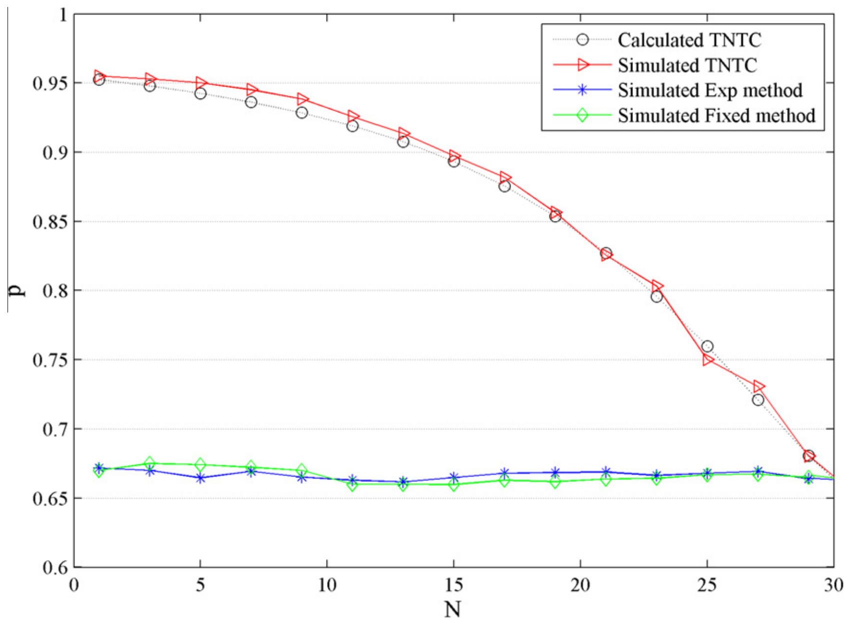

Fig. 14. Comparison of *p* with varying *N* between TNTC and timer methods.

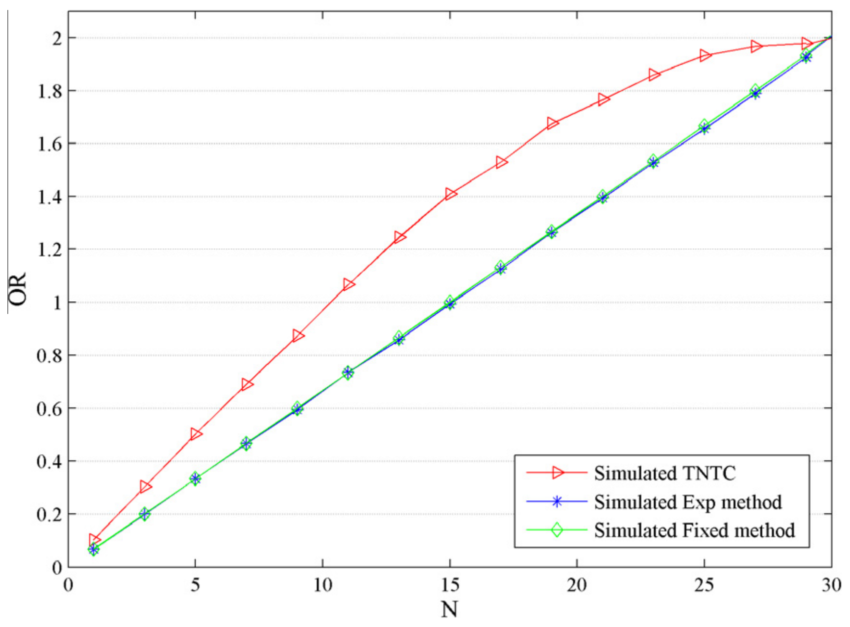

Fig. 15. Comparison of *OR* with varying *N* between TNTC and timer methods.

deal with higher update arrival rate and provide higher valid access probability than timer methods.

5. Conclusion and future work

In this paper, we proposed TNTC, which is a token-bucket based notification traffic control mechanism for IMS presence service. TNTC aims at upgrading the valid access probability while controlling the notification traffic. We firstly presented the design of the proposed approach, then modeled the TNTC using queuing theory and calculated its main probability features. Based on the analytical modeling, we investigated the effect of different parameters on the performance of TNTC, and presented a guideline about the settings of these parameters. Finally, we compared the performance of TNTC with Fixed and Exp methods through simulations. Simulation results show that TNTC outperforms other methods in terms of the valid access probability and the update arrival rate.

In our future work, the selection of appropriate value for parameter c needs to be further studied. Besides, as the presence-based enterprise services for inter-domain enterprise mobility are now receiving more and more attentions [22], we will explore the distributed notification traffic control mechanism among multiple presence servers from different domains to ensure that the bandwidth resources are rationally utilized. In addition, we will investigate the differentiated service in presence service, which selects the different notification traffic control policy depending on the user class to improve the QoS of presence service.

Acknowledgements

This work was jointly supported by National Science Fund for Distinguished Young Scholars (No. 60525110); National 973 Program (No. 2007CB307100, 2007CB307103); National Natural Science Foundation of China (No. 61072057, 60902051); the Fundamental Research Funds for the Central Universities (BUPT2009RC0505); Development Fund Project for Electronic and Information Industry (Mobile Service and Application System Based on 3G); National Key Science & Technology Specific Project of China (No. 2011ZX03002-001-01, 2011ZX03002-002-01). MICINN (No. TIN2010-19077); CAM (No. S2009TIC-1692).

Appendix A

In order to further explain (14), this appendix shows the deduction from [25].

Let $C(t,t+h)$ be the event that a message arrives in the interval $(t,t+h)$, and define

$$\pi_k^{[s]}(t) = \lim_{h\to 0} P\{S(t)=k|C(t,t+h)\} \tag{A.1}$$

Because $\{S(t), t \geqslant 0\}$ is the birth–death process, we have:

$$\lim_{h\to 0} P\{C(t,t+h)|S(t)=k\} = \lambda_k h + o(h) \tag{A.2}$$

For the purpose of making calculations for our system, we express (A.1) in the form commonly referred to as Bayes's rule:

$$\pi_k^{[s]}(t) = \lim_{h\to 0} \frac{P\{C(t,t+h)|S(t)=k\}P_k^{[s]}(t)}{\sum_{j=0}^{c+s} P\{C(t,t+h)|S(t)=j\}P_j^{[s]}(t)} \tag{A.3}$$

Substitute (A.2) into (A.3), and simplify the result, we can write:

$$\pi_k^{[s]}(t) = \frac{\lambda_k P_k^{[s]}(t)}{\sum_{j=0}^{c+s}\lambda_j P_j^{[s]}(t)} \tag{A.4}$$

Substituting (5) and (13) into (A.4), we conclude that:

$$\pi_k^{[s]} = \lim_{t\to\infty}\pi_k^{[s]}(t) = \lim_{t\to\infty}\frac{\lambda_k P_k^{[s]}(t)}{\sum_{j=0}^{c+s}\lambda_j P_j^{[s]}(t)} = \frac{\lambda_k P_k^{[s]}}{\sum_{j=0}^{c+s}\lambda_j P_j^{[s]}} \tag{A.5}$$

Appendix B

This appendix deduces the waiting time distribution of arrival PUBLISH message on the condition that the number of waiting messages in viewpoint of this arrival message is i.

A PUBLISH message arrives and finds there is no token left in token bucket and i other messages waiting in the queue. Messages are served in order of arrival, and all messages wait in the queue till a token is obtained. Let X_1 be the elapsed time from $t = 0$ until the message at the head of the queue obtains a token; and let X_j ($j = 2, 3, \ldots, i+1$) be the length of time that the jth message spends at the head of the queue (X_{i+1} is the time that arrival message spends at the head of the queue). Clearly, the arrival message's waiting time is the sum $X_1 + X_2 + \cdots + X_{i+1}$. Because the token generating time interval is exponential distributed with mean u^{-1} as shown in Section 3.2, $X_1, X_2, \ldots, X_{i+1}$ are all independent, identical and exponential variables with mean u^{-1}. Thus, we conclude that the waiting time T_w of arrival message, given that the number of messages it finds waiting ahead of it in the queue is i, has the $(i+1)$-phase Erlangian distribution [25]:

$$P\{T_w > t|Q=i\} = P\{X_1+X_2+\cdots+X_{i+1} > t\} = \sum_{j=0}^{i}\frac{(ut)^j}{j!}e^{-ut} \tag{B.1}$$

Appendix C

In this appendix, we give the deduction of the Laplace transform of probability density function of waiting time as shown in (27). The waiting time T_w has probability density function $f(t)$ with its corresponding Laplace transform $f^*(s')$, and cumulative distribution function $F(t)$ with Laplace transform $F^*(s')$. Let $\mu(t)$ denote unit step function and $L[\,]$ denote the operation of Laplace transform, that is $L[f(t)] = f^*(s')$. From (26), we can write:

$$F(t) = 1 - \sum_{i=0}^{s-1}\left(\pi_{c+i}^{[s]}\sum_{j=0}^{i}\frac{(ut)^j}{j!}e^{-ut}\right) \tag{C.1}$$

Considering $t > 0$, the Laplace transform of cumulative distribution function is:

$$\begin{aligned} F^*(s') = L[F(t)] &= L[\mu(t)] - \sum_{i=0}^{s-1}\left(\pi_{c+i}^{[s]}\sum_{j=0}^{i}\frac{u^j}{j!}L[t^j e^{-ut}]\right) \\ &= \frac{1}{s'} - \sum_{i=0}^{s-1}\left(\pi_{c+i}^{[s]}\sum_{j=0}^{i}\frac{u^j}{j!}\frac{j!}{(s'+u)^{j+1}}\right) \\ &= \frac{1}{s'} - \sum_{i=0}^{s-1}\left(\pi_{c+i}^{[s]}\sum_{j=0}^{i}\frac{u^j}{(s'+u)^{j+1}}\right) \end{aligned} \tag{C.2}$$

By the theorem of Laplace transform as shown:

$$L[f(t)] = L\left[\frac{dF(t)}{dt}\right] = s'F^*(s') - F(0) \tag{C.3}$$

Substitute (C.2) into (C.3) and consider $F(0) = 1$ from (C.1), we can get the conclusion:

$$f^*(s') = L[f(t)] = -s'\sum_{i=0}^{s-1}\left(\pi_{c+i}^{[s]}\sum_{j=0}^{i}\frac{u^j}{(s'+u)^{j+1}}\right) \tag{C.4}$$

References

[1] C. Urrutia-Valds, A. Mukhopadhyay, M. El-Sayed, Presence and availability with IMS: applications architecture, traffic analysis, and capacity impacts, Bell Labs Technical Journal 10 (4) (2006) 101–107.
[2] Y. Zhang, J. Liao, X. Zhu, W. Wu, J. Ma, Inter-working between SIMPLE and IMPS, Computer Standards & Interfaces 29 (5) (2007) 584–600.
[3] 3GPP TS 22.141, Presence service; Stage 1; release 7, December 2005.
[4] 3GPP TS 23.141, Presence service; Architecture and functional description; Stage 2; Release 8, June 2008.
[5] M. Day, J. Rosenberg, H. Sugano, A model for presence and instant messaging, IETF RFC 2778, February 2000.
[6] 3GPP TR 24.841, Presence service based on Session Initiation Protocol (SIP); Functional models, information flows and protocol details; Release 6, June 2004.
[7] M. Day, S. Aggarwal, G. Mohr, J. Vincent, Instant Messaging/Presence Protocol Requirements, IETF RFC 2779, February 2000.
[8] F. Wegscheider, Minimizing Unnecessary Notification Traffic in the IMS Presence System, in: Proceedings of ISWPC2006, Phuket, Thailand, January 2006.
[9] Muhanmmad T. Alam, Z. Wu, Cost analysis of the IMS presence service, in: 1st Australian Conference on Wireless Broadband and Ultra Wideband Communication, AusWireless 2006, Sydney, March 2006.
[10] M. Pous, D. Pesch, G. Foster, A. Sesmun, Performance evaluation of a SIP based presence and instant messaging service for UMTS, in: 4th International Conference on 3G Mobile Communication Technologies, London, UK, June 2003, pp. 254–258.
[11] Muhanmmad T. Alam, Z. Wu, Admission control approaches in the IMS presence service, International Journal of Computer Science 1 (4) (2006) 299–314.
[12] Z. Cao, C. Chi, R. Hao, Y. Xiao, User behavior modeling and traffic analysis of IMS presence service, in: Proceedings of GLOBECOM2008, New Orleans, LA, USA, November 2008, pp. 1–5.
[13] C. Chi, R. Hao, D. Wang, Z. Cao, IMS presence server: traffic analysis & performance modeling, in: Proceedings of ICNP2008, Orlando, Florida, USA, October 2008, pp. 63–72.
[14] J. Rosenberg, A presence event package for the Session Initiation Protocol (SIP), IETF RFC 3856, August 2004.
[15] W.-E. Chen, Y.-B. Lin, R.-H. Liou, A weakly consistent scheme for IMS presence service, IEEE Transactions on Wireless Communications 8 (7) (2009) 3815–3821.
[16] A.B. Roach, A Session Initiation Protocol (SIP) event notification extension for resource lists, IETF RFC 4662, August 2006.
[17] M. Lonnfors, J. Costa-Requena, E. Leppanen, H. Khartabil, Session Initiation Protocol (SIP) extension for partial notification of presence information, IETF RFC 5263, September 2008.
[18] H. Hannu, Signaling compression (SigComp) requirements & assumptions, IETF RFC 3322, January 2003.
[19] D. Pesch, M.I. Pous, G. Foster, Performance evaluation of SIP-based multimedia services in UMTS, Computer Networks 49 (3) (2005) 385–403.
[20] V.K. Singh et al., Presence traffic optimization techniques, Technical Report, Columbia University, October 2006.
[21] S. Loreto, G.A. Eriksson, Presence network agent: a simple way to improve the presence service, IEEE Communications Magazine 46 (8) (2008) 75–79.
[22] P. Bellavista, A. Corradi, L. Foschini, IMS-based presence service with enhanced scalability and guaranteed QoS for interdomain enterprise mobility, IEEE Wireless Communications 16 (3) (2009) 16–23.
[23] A. Niemi, K. Kiss, S. Loreto, Session Initiation Protocol (SIP) event notification extension for notification rate control, internet draft-ietf-sipcore-event-rate-control-07, April 2011.
[24] Andrew S. Tanenbaum, Computer Networks, fourth ed., Prentice-Hall, 2003. pp. 397–417.
[25] Robert B. Cooper, Introduction to Queueing Theory, Elsevier North Holland, Inc., 1981. pp. 42–98.
[26] L. Kleinrock, Queueing Systems, Theory, vol. I, John Wiley & Sons, New York, 1975. pp. 10–94.
[27] W.R. LePage, Complex Variables and the Laplace Transform for Engineers, Dover, New York, 1980. pp. 285–328.
[28] SIMPROCESS simulator. <http://simprocess.com/>.