Modeling and design of session initiation protocol overload control algorithm

Yang Hong, Changcheng Huang, James Yan
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
E-mail: {yanghong, huang}@sce.carleton.ca, jim.yan@sympatico.ca

(*received 28 September 2011; revised 21 December 2011; accepted 3 March 2012*)

Abstract
Recent collapses of Session Initiation Protocol (SIP) servers indicate that the built-in SIP overload control mechanism cannot mitigate overload effectively. In this paper, we propose a new SIP overload control algorithm by introducing a novel analytical approach to model the dynamic behaviour of a SIP network where each server has a finite buffer. Three key breakthroughs of our modeling approach are the formulations of message loss process, message retransmission process, and the complex departure process through detailed analysis. Our modeling results indicate that retransmissions triggered by the queuing delay are redundant, thus we propose a feedback control mechanism that regulates retransmission message rate to mitigate the overload. We then demonstrate how to extend our analytical approach to the modelling of our overload control solution. Simulation based on this analytical model runs much faster than event-driven simulation which needs to track thousands of retransmission timers for outstanding messages and may crash a simulator due to limited computation resources. Performance evaluation demonstrates that: (1) without the control algorithm applied, the overload at a downstream server may propagate to its upstream servers and cause widespread network failure; (2) in case of short-term overload, our feedback control solution can mitigate the overload effectively without rejecting calls intentionally or reducing network utilization, thus avoiding the disadvantages of existing overload control solutions. In addition, compared with the pushback solution, our retransmission-based solution achieves a better trade-off between the speed to cancel the overload and the call rejection rate when an overload lasts a short period.

Keywords
CPU Utilization, Non-redundant Retransmission, Overload Control, SIP, SIP Retransmission.

1. Introduction

Internet telephony is experiencing rapidly growing deployment due to its lower-cost telecommunications solutions for both consumer and business services. Session Initiation Protocol (SIP) [1] has become the main signaling protocol to manage multimedia sessions for numerous Internet telephony applications such as Voice-over-IP, instant messaging and video conferencing. 3rd Generation Partnership Project (3GPP) has adopted SIP as the basis of its IP Multimedia Subsystem (IMS) architecture [2]. With the 3rd Generation (3G) wireless technology being adopted by more and more carriers, most cellular phones and other mobile devices are starting to use or are in the process of supporting SIP for multimedia session establishment [3].

RFC 5390 [4] identified the various reasons that may cause server overload in a SIP network. These include but not limited to poor capacity planning, dependency failures, component failures, avalanche restart, flash crowds, etc. In general, anything that may trigger a demand burst or a server slowdown can cause server overload and lead to server crash. There are many published works [5-31] discussing how to control SIP overload through call rejection or load balancing. These mechanisms can either increase call rejection rate or cause SIP server underutilized. Rejecting calls can cancel the overload effectively, but it may cause a large amount of revenue loss to carriers. We believe that arbitrarily high volume of call rejection can be avoided in most overload situations if we can understand the root cause of widespread network failures by modeling server overloading behaviour before designing appropriate overload control algorithm.

Traffic modeling has played a large impact on network simulation reliability in the past years [32]. Modeling and simulation can help enterprise engineers to optimize the planning and the dimensioning of their communication network or system [33]. Numerous models for different communication system have been developed recently, e.g., a model was proposed by [32] to mimic packet defragmentation process based on the measured network traffic. In order to find the optimum number of input connections required for serving a certain quota of users, the paper [33] presented a model for predicting the system capacity of accepting the calls into a server.

The objective of this paper is to develop a new overload control algorithm through analytical modeling approach that allows us to investigate how server overloading and widespread SIP network failure may happen under short term demand bursts or server slowdowns. We want to demonstrate that the major cause of this kind of failures is the hop-by-hop retransmission mechanism designed for reliability purpose in SIP. Different from existing overload control solutions that try to mitigate overload through rejecting calls or load balancing, our approach will address redundant retransmissions directly. This is particularly useful when the overload is caused by short term demand bursts or temporary server slow down. For overloading caused by persistent demand surge or server slowdown, our solution can be used together with existing solutions to maximize server utilization.

The contributions of this paper are: (1) Creating an innovative approach to formulate analytical models for SIP server networks with finite buffer sizes using difference equations. The arrival and service processes of each SIP server can be arbitrary, which makes our modeling approach quite general. We have successfully analyzed different types of message losses, retransmission messages triggered by queuing delay or message drops, and formulated the complex departure processes. These analytical results allow us to understand the impacts of redundant retransmissions and finite buffer sizes; (2) Developing a novel overload control algorithm to mitigate the overload by controlling retransmission probability based on the average CPU utilization of the overloaded downstream server. In addition, we apply our analytical modeling approach to show that the proposed overload control mechanism is effective in preventing overload collapse under two typical scenarios of short-term overload. For a short-term overload, our retransmission-based solution achieves a better trade-off between the speed to cancel the overload and the call rejection rate when compared with the pushback solution; (3) Demonstrating the efficiency and accuracy of our analytical model through a performance comparison between fluid-based Matlab simulation and event-driven OPNET simulation of a SIP network.

This study will help network planners, operators, and researchers to understand the root cause of an important network failure scenario triggered by short term traffic surges such as flash crowds and database maintenance. Network planners can therefore plan their SIP networks better by engineering their SIP networks to avoid redundant retransmissions. Operators can apply our feedback control mechanism to mitigate the short-term overload and maintain the revenue without rejecting calls intentionally. Researchers can use our model to speed up the performance evaluation of various SIP overload control solutions using the fluid-based simulation, when a SIP network is scaled up.

The paper is organized as follows. Section 2 reviews existing overload control solutions and simulation approaches. Section 3 briefly reviews SIP protocol. Section 4 proposes our analytical modeling approach for SIP networks. The overload control algorithm is developed and corresponding analytical model is created in Section 5. Section 6 evaluates performance of a SIP network under two typical overload scenarios via Matlab simulation and OPNET simulation. Conclusions are given in Section 7.

2. Related work

2.1. SIP overload control

Experimental evaluation of SIP server showed the overload collapse behaviour in [34]. Recent collapses of SIP servers due to emergency induced call volume or “American Idol” flash crowd in real carrier networks have attracted great research attention and motivated different types of strategies to address SIP server overload problem (e.g., [5-29]).

1) Load Balancing: As a default part of numerous operating systems, SIP Express Router (SER) developed load balancing module to mitigate the overload caused by large subscriber populations or abnormal operational conditions [8]. In order to balance the demands for bandwidth and the call failure rates of a SIP proxy, request batching was combined with parallel execution to improve call throughput and reduce call failure rate significantly in [9]. Three novel approaches were introduced for load balancing in cluster-based SIP servers [10]. The load balancer performed session-aware request assignment to route SIP transactions to the proper back-end nodes [10]. Peer-to-Peer network technology was integrated with SIP to balance the traffic load (e.g., [11, 12]).

In an IMS core network, a load balancing scheme was proposed to reduce overload probability by re-directing consequent SIP traffic from the over-utilized Serving Call/Session Control Function (S-CSCF) server to the other under-utilized ones [13]. A SIP message overload transfer scheme can leverage redundant Interrogating CSCF (I-CSCF) servers to reduce the message disruption in cases of server failures [14]. A user equipment registration scheme not only balanced the workload over multiple Proxy-CSCF (P-CSCF) nodes, but also reduced the required P-CSCF nodes up to 40% from the standard session initialization procedure of IMS [15].

However, load balancing tries to avoid SIP network failures by reducing the utilization of those servers that may become overloaded. This will increase network cost and therefore reduce revenue. When the total message arrival rate exceeds the aggregated processing capacities of all local servers, load balancing schemes cannot prevent the overload collapse.

2) Priority-Based Overload Control: Before finding its application in preventing SIP overload due to CPU constraint, priority mechanism has been adopted for active queue management due to bandwidth constraint, e.g., stateless fair admission control (SFAC) aims at guaranteeing fair bandwidth allocation for each flow [35]. A priority enqueuing scheme provided differentiate service for different types of SIP messages in every SIP proxy server, where INVITE requests were placed into low priority queue [16]. Once the proxy server was overloaded, every INVITE request was hardly forwarded to its destination, thus forbidding the successive non-INVITE transactions to reduce the traffic load [16]. Another queuing strategy applied a queue threshold for the INVITE request message queue to detect the overload [17]. When the overload was anticipated, part of INVITE requests would be rejected to mitigate the overload, and the corresponding calls were blocked [17]. Priority queue was used to overcome the overload problem of IMS system by blocking non-priority calls [18]. Similar to the priority scheme, a novel authentication protocol was developed to reduce the load on the centralized authentication database dramatically and improve the overall security of a carrier-scale VoIP network [19].

3) Pushback Overload Control: Since the cost of rejecting a session intentionally is usually comparable to the cost of serving a session [21], cancelling INVITE transaction using priority queuing scheme is not very cost effective. Therefore, numerous pushback solutions have been proposed to reduce the traffic loads of an overloaded receiving server by advertising its upstream sending servers to decrease their sending rates.

Both centralized and distributed overload control mechanisms for SIP were developed in [7]. Retry-after control, processor occupancy control, queuing-delay control and window-based control were proposed to improve goodput and prevent overload collapse in [20]. Three window-based feedback algorithms were proposed to adjust the message sending rate of the upstream SIP servers based on the queue length [21]. Other pushback solutions can be found in [5, 22-28].

Such pushback control solutions aim at preventing the overload of a server by reducing the sending rate of its upstream servers. This would increase the queuing delays of newly arrival original messages at the upstream servers, which in turn cause overload at the upstream servers. Overload may thus propagate server-by-server to sources. Unlike TCP situation where a source typically generates large amount of data, a UAC in SIP only generates very few signalling messages [7]. Pushback solution leads to rejection of a large number of calls which means revenue loss for carriers. However, it may be unnecessary to reject calls intentionally when temporary overload only lasts a short period of time.

4) Proposal of Retransmission-Based Overload Control: Our approach proposed in this paper is different from the existing solutions discussed above. When retransmissions are caused by the overload rather than the message loss, they will bring extra overhead instead of reliability to the network and exacerbate the overload as we will demonstrate later through our analytical modeling and simulation. Therefore, when a short-term overload occurs at a server, we propose to reduce the retransmission rate instead of reducing original message sending rate of the upstream servers.

Such retransmission-based solution will mitigate the overload while maintaining original message sending rate, which leads to less blocking calls and more revenue for carriers. The key to this novel solution is to differentiate necessary retransmissions from redundant ones. Direct or indirect ways may be used to address this goal. With indirect approaches, the upstream servers can try to guess whether downstream servers are overloaded through differences in delays of response messages [36-39]. Although this kind of control mechanisms does not require any modification to SIP protocol, it unfortunately might lead to overreaction and potential throughput loss. Therefore, we propose in this paper to control retransmission rate based on explicit indication from downstream server. Analytical and simulation results shown later indicate that by introducing a minor change to SIP protocol, our new control mechanism eliminates the short-term overload more quickly.

When an overload lasts for a long period, our retransmission-based solution can be combined with pushback solution to reject some calls by reducing the original message rates of SIP sources. Under this kind of situation, our solution can help maximize server utilization and therefore serve as a complementary mechanism to the existing solutions.

2.2. Fluid-based simulation vs. event-driven simulation

Fluid-based simulation was originally created to speed up simulation processes [40]. Before fluid-based simulation was introduced, event-driven simulation had been widely used for network performance evaluation. Its computation cost grows linearly with network size and bandwidth [40]. Therefore, it is necessary to develop some other approaches to simplify the event-driven simulation. Fluid-based simulations have been successfully used to achieve scalability by aggregating events into time slots [40]. However, fluid-based simulation is highly dependent on whether analytical models can be established to capture the dynamic process of a system. The retransmission mechanism of TCP was not modelled in [40] due to the complexity of modeling retransmission mechanism.

In this paper, our goal is to develop an analytical model that can help us understand how the widespread failure of a SIP network may happen due to the SIP retransmission mechanism so that we can design a simple and effective overload control mechanism. It should be noted that it is difficult to apply the fluid-based simulations in [40] directly to SIP because a large number of variable timers in SIP generate complex correlation structures at different time scales. In order to solve this problem, we have to study the message loss process, message retransmission process, and service departure process in detail by introducing some new techniques which will be discussed in later sections.

Our analytical model can also be used to conduct fluid-based simulation and speed up simulation process as [40]. This is another important benefit of our model. We will demonstrate that our fluid-based simulation can speed up simulation process significantly compared with event-driven approach, when a SIP network has to process a large number of signaling messages.

We have investigated the chaotic behaviour of SIP retransmission mechanism by assuming that each SIP server has infinite buffer and no message will be dropped due to buffer overflow, while no any SIP overload control algorithm is activated [41, 42]. In this paper, each SIP server has finite buffer, and messages will be dropped when the buffer becomes full. In addition, an analytical model of SIP overload control algorithm is also developed.

3. SIP Protocol overview

Figure 1 illustrates a basic operation of a SIP system. To set up a call, a user agent client (UAC) sends an INVITE request to a user agent server (UAS) via the two proxy servers. The proxy server returns a provisional 100 (Trying) response to confirm the receipt of the INVITE request. The user agent server returns a 180 (Ringing) response after confirming that the parameters are appropriate. It also evicts a 200 (OK) message to answer the call. The user agent client sends an ACK response to the user agent server after receiving the 200 (OK) message. Finally the call session is established between the user agent client and the user agent server through the SIP session. The BYE request is generated to close the session thus terminating the communication. When a SIP proxy server is overloaded, it will send a 503 Service Unavailable message in response to an INVITE request. The call will then be rejected. Many existing pushback SIP overload control proposals are based on the 503 Service Unavailable messages to reject calls when servers are overloaded as discussed in the previous section. Processing SIP INVITE requests and generating 503 Service Unavailable messages still consume a large amount of CPU time. Therefore it is questionable that the server load can be actually reduced through this kind of approach.

As the INVITE message is the most complex message to be processed by a SIP server and thus the major CPU load contributor [1], we will focus on the INVITE-100Trying transaction and ignore other non-INVITE transactions in this paper. Given the proportionate nature and the general similarity of the retransmission mechanisms between the INVITE and non-INVITE messages in a typical session [1], our modeling approach can be naturally extended to include non-INVITE transactions.

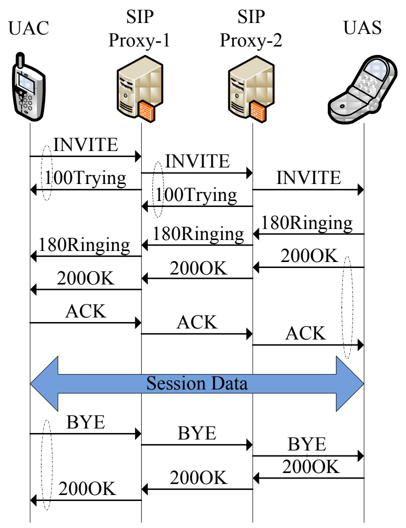


Figure 1. A typical procedure of session establishment.

SIP introduces a retransmission mechanism to maintain its reliability [1, 43]. In practice, a SIP sender uses timeout to detect message losses. One or more retransmissions would be triggered if the corresponding reply message is not received in predetermined time intervals.

SIP RFC 3261 [1] suggests that the SIP retransmission mechanism should be disabled for hop-by-hop transaction when running SIP over TCP to avoid redundant retransmissions at both SIP and TCP layer. However, SIP retransmission mechanism is mandatory for end-to-end transactions no matter TCP or UDP is used [1].

Recent experimental evaluation on SIP-over-TCP overload behaviour in [5] demonstrates that TCP flow control mechanism cannot prevent SIP overload collapse for time-critical session-based applications due to lack of application context awareness at the transport layer. Other experiments (e.g., [6, 7]) also indicate that the throughput with SIP-over-TCP exhibits similar overload collapse behaviour as that with SIP-over-UDP. In addition, experiment performed by IBM research center in [34] claims that using TCP to deliver SIP messages degrades server performance from 43% (under stateful proxy with authentication) to 65% (under stateless proxy without authentication) when compared with using UDP. Therefore, running SIP over UDP is a good option for vendors in practice.

The retransmission for an INVITE transaction is confirmed on a hop-by-hop basis. For each hop, the sender starts the first retransmission of the original message at T_1 seconds, and the time interval doubles after every retransmission (exponential back-off), if the corresponding reply message is not received. The last retransmission is sent out at the maximum time interval $64\times T_1$ seconds. Thus there is a maximum of 6 retransmissions. The default value of T_1 is 0.5s [1].

To select a topology that can be considered typical, we first consider the different types of SIP nodes. There are in general two types of SIP nodes as shown in Figure 1: UAs vs. proxy servers. In some cases, application servers can take the place of UAs. In the following study, we will start with a tandem server topology to cover both types of nodes, as shown in Figure 2. Server 1 represents an arbitrary proxy server, while Server 2 represents an arbitrary UA. Therefore, we consider that Server 1 receives the responses from downstream nodes while Server 2 does not. The original messages and the retransmitted messages arriving at Server 1 are merged aggregate streams from multiple upstream servers. If there are more proxy servers between Server 1 and Server 2, their behaviour patterns will be similar to Server 1 in our tandem server scenario.

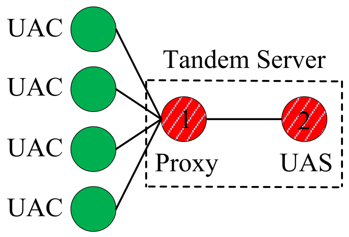


Figure 2. SIP network topology with an overloaded tandem server which is marked with diagonal lines.

After finishing the study on the tandem server, we will discuss the impact of splitting the output of Server 1 to multiple downstream servers and merging traffic from multiple upstream servers at Server 2. This will allow us to generalize our results to arbitrary topology in Section 4.

We make the following statement in accordance with SIP RFC 3261 [1] and the requirements for practical SIP networks without losing the generality of our approach:

(a) Delay Consideration: We investigate the retransmissions which are mainly caused by long queuing delay of the overloaded server. Therefore, for the round trip response time between an overloaded server and its neighbouring server, the queuing and processing delays are dominant, while transmission and propagation delay are negligible [21]. This consideration is valid because signaling messages are typically CPU capacity constrained rather than bandwidth constrained;

(b) Discrete-time Formulation: Time is divided into discrete time slots. This allows us to develop discrete time models which are much easier to understand and simulate. It is easy to see that the errors caused by the discrete model can be made arbitrarily small by making the interval of a timeslot smaller and smaller. In this paper, in order to present our analysis clearly, we use t and n to denote time and timeslot respectively. The default interval of a time slot is 50ms and the corresponding values of T_1 are 0.5s or 10 timeslots respectively;

(c) Message Enqueuing/Dropping Policy: The SIP RFC 3261 [1] does not specify the queuing and scheduling discipline to be deployed by a SIP server. Without loss of the generality of our approach, we let a SIP server maintain a first-come-first-served (FCFS) queue for messages arriving at different time slots. When a buffer is full, traffic arrivals are dropped randomly. Different types of traffic arrivals, such as original request messages, retransmitted request messages from upstream servers, retransmitted messages for the downstream servers, etc., are therefore dropped proportionally according to their corresponding amounts of traffic arrivals. When the buffer is not full, traffic arrivals in different time slots will follow first-come-first-in policy consistent with the FCFS principle. However, for the simplicity of mathematical treatment, traffic arrivals within the same time slot will be enqueued based on a priority scheme to be described in the next section. The impact of this priority scheme within a timeslot can be made arbitrarily small when the interval of the time slot decreases;

(d) Response Treatment: Enqueuing response messages at the tail of the message queue will delay the processing of response messages, thus trigger more redundant retransmissions and make the overload worse. We assume response messages be handled as interrupts and enter the head of message queue if cannot be processed immediately. It should be noted that the time to process a response message is typically much smaller than a request message;

(e) Overload Assumption: When overload happens in the network, at any time, one of the servers will be the most congested one among all the overloaded servers. It becomes the bottleneck server. We let the tandem server be the bottleneck server by making capacities of all its upstream servers large enough to process the messages without any delay. The tandem server consists of Server 1 and Server 2 (as shown by Figures 2 and 3). Both Server 1 and Server 2 have limited capacities.

4. Modeling a SIP network

In order to understand the impact of retransmission mechanism, we need to analyze the queuing and retransmission processes in details. This motivates us to develop an analytical model for a general SIP network. To provide a better understanding of our modeling approach, we create an analytical model for a tandem server first. Then we discuss how to extend our modeling approach to an arbitrary network with minor modifications.

There are three main challenges for analyzing SIP servers with finite buffer: (1) the different types of arrival messages a SIP server has to process; (2) the impact of different types of message drops on the SIP retransmissions due to the buffer overflow; (3) the complex relations among arrival process, service process, retransmission process, response process and queuing process, e.g., arrival retransmitted messages and arrival response messages depend on the queuing and departure processes of both upstream servers and downstream servers as discussed later on.

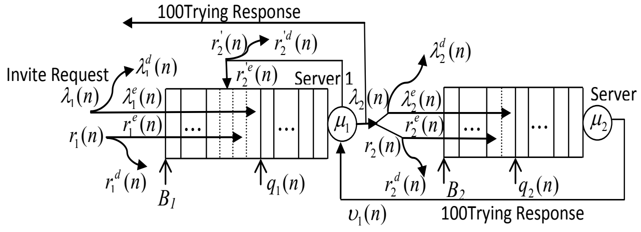


Figure 3. Queuing dynamics of a tandem SIP server (*For Server 1*, $\lambda_1(n)$ denotes original message arrivals, $r_1(n)$ denotes retransmitted message arrivals from upstream servers, $r'_2(n)$ denotes retransmitted messages created for Server 2, $q_1(n)$ denotes queue size, $\mu_1(n)$ denotes service rate, B_1 denotes finite buffer size; *For Server 2*, $\lambda_2(n)$ denotes original message arrivals, $r_2(n)$ denotes retransmitted message arrivals, $q_2(n)$ denotes queue size, $\mu_2(n)$ denotes service rate, B_2 denotes finite buffer size).

We start with a tandem server to introduce our fundamental modeling approach to address the three challenges. Figure 3 depicts the queuing dynamics of a tandem server, where Server 2 is an arbitrary user agent server and Server 1 is an arbitrary proxy server. According to Overload Assumption (e), both servers are assumed to have limited capacity and therefore are prone to overload. Server 2, as a user agent, does not need to consider response messages from downstream servers.

Different types of messages have different processing requirements and are typically treated differently by SIP servers. Our approach tries to differentiate different types of messages into three categories and formulate their processing and departure differently. The three types of arrival messages of Server 1 are: original INVITE request arrival process $\lambda_1(n)$ from upstream servers, retransmitted INVITE request arrival process $r_1(n)$ from upstream server, and response arrival process $\upsilon_1(n)$ from downstream server which depends on message departure process of upstream server.

In order to determine whether an original message needs to be retransmitted by an upstream server for Server 1, according to SIP RFC [1], we need to know whether the message is still in queue at the Server 1 at current time or it has been dropped by Server 1 due to buffer overflow. Different types of message drops would generate different retransmission scenarios, because the drop of redundant retransmitted messages will not trigger new retransmissions, while the drop of non-redundant retransmitted messages may trigger new retransmissions.

The arrival process, the departure process and service process are related, while retransmission and response depend on the queuing behaviour of both upstream server and downstream server. Both original message arrival process $\lambda_2(n)$ of Server 2 and retransmission process $r_1(n)$ of Server 1, are determined by original message arrival process $\lambda_1(n)$ of Server 1 and the queuing dynamics of Server 1. Response process $\upsilon_1(n)$ of Server 1 and retransmission process $r_2(n)$ of Server 2, are determined by original message arrival process $\lambda_2(n)$ of Server 2 and the queuing dynamics of Server 2. The arrival processes at Server 2 including the original INVITE request messages λ_2(n) and retransmitted request messages $r_2(n)$ depend on the departure process of Server 1. The queuing dynamics of both Server 1 and Server 2 are determined by original message arrival process, service process retransmission process and response process.

Given that both original message arrival process $\lambda_1(n)$ of Server 1, server service process $\mu_1(n)$ of Server 1, and server service process $\mu_2(n)$ of Server 2 can be arbitrarily distributed, the relation between all different dynamic processes is not trivial. Thus obtaining retransmission process $r_1(n)$ and $r_2(n)$, departure process $\lambda_2(n)$ and $\upsilon_1(n)$ is a difficult task. Since they all depend on the queuing processes at both servers (as shown by Figure 3), we start with the queuing processes for our modeling.

We consider the Server 2 first. Since Server 2 has finite buffer size, the new arriving messages (including original and retransmitted messages) are dropped when the buffer is full (i.e., the queue size $q_2(n)=B_2$, where B_2 denotes the finite buffer size of Server 2, as shown in Figure 3). At the current time slot n, the original INVITE requests arrive with an aggregate rate $\lambda_2(n)$: only $\lambda_2^e(n)$, which are called effective original arriving messages, enter the buffer of Server 2 and $\lambda_2^d(n)$ are dropped; similarly the retransmitted INVITE requests arrive with an aggregate rate $r_2(n)$: only $r_2^e(n)$, which are called effective retransmitted message arrivals, enter the buffer of Server 2 and $r_2^d(n)$ are dropped; Server 2 can process $\mu_2(n)$ messages and $\mu_2(n)$ can be an arbitrary stochastic process.

Similarly, for Server 1, the queue size is $q_1(n)$ and the finite buffer size is B_1. At current time slot n, the original INVITE requests (from all the upstream servers of Server 1) arrive with an aggregate rate $\lambda_1(n)$: only $\lambda_1^e(n)$ enter the buffer and $\lambda_1^d(n)$ are dropped; the retransmitted INVITE requests arrive with an aggregate rate $r_1(n)$, where $r_1(n)$ denotes the retransmitted messages: only $r_1^e(n)$ enter the buffer and $r_1^d(n)$ are dropped; Server 1 generates the retransmitted messages for $\lambda_2(n)$ with a rate $r'_2(n)$ which also need to go through Server 1: only $r'^e_2(n)$ enter the buffer and $r'^d_2(n)$ are dropped; the response messages from Server 2 corresponding to $\lambda_2(n)$ arrive at Server 1 with a rate $\upsilon_1(n)$; Server 1 can process $\mu_1(n)$ messages. Since the processing time for different type of messages may be different, $\mu_1(n)$ denotes an arbitrary and generic process. Based on Enqueuing Policy (c), $r'^e_2(n)$ is not equal to $r_2(n)$ due to queuing and processing delays at Server 1.

We derive the queuing dynamics of Server 1 first. Assuming $\lambda_1(n)$, $\mu_1(n)$, and B_1 given, our goal is to calculate $\lambda_1^e(n)$, $\lambda_1^d(n)$, $r_1^e(n)$, $r_1^d(n)$, $r'^e_2(n)$, $r'^d_2(n)$, $\upsilon_1(n)$ and $q_1(n)$. Clearly the key issue to be addressed is how to know which messages are actually lost. It is a difficult and challenging issue because message losses are random in nature. They can happen to original messages and retransmitted messages. With arbitrary message arrival processes and server serving processes, the issue becomes even harder to solve. In the following, we will show how we successfully solve the problem in a step-by-step manner.

At the beginning of current time slot n, the available buffer size will be $B_1-q_1(n)$, and the server can process $\mu_1(n)$ messages in the slot. Since Server 1 has enough CPU capacity to process the incoming response messages (i.e., Response Treatment (d)), the buffer only allows maximum $B_1-q_1(n)-\upsilon_1(n)+\mu_1(n)$ new arrival request messages to enter the queue. If the total arrival request messages exceed the available processing capacity, the buffer overflow happens and the request messages are dropped randomly. Within a timeslot, messages are dropped proportionally according to their corresponding amounts of traffic arrivals (i.e., Dropping Policy (c)). Thus the drop probability p_{d1} can be obtained as

$$p_{d1}(n)=\left[\frac{\lambda_1(n)+r_1(n)+r'_2(n)+\upsilon_1(n)+q_1(n)-B_1-\mu_1(n)}{\lambda_1(n)+r_1(n)+r'_2(n)}\right]^+ \quad (1)$$

where use $[]^+$ to denote nonnegative value.

Then the original messages $\lambda_1(n)$ enter the buffer with the probability $(1-p_{d1})$. We can obtain $\lambda_1^e(n)$ as

$$\lambda_1^e(n)=\lambda_1(n)(1-p_{d1}(n)). \quad (2)$$

If the buffer overflow happens, then the extra messages $\lambda_1^d(n)$ are dropped. So we can obtain $\lambda_1^d(n)$ as

$$\lambda_1^d(n)=\lambda_1(n)-\lambda_1^e(n). \quad (3)$$

Similarly we can obtain $r_1^e(n)$, $r_1^d(n)$, $r'^e_2(n)$, and $r'^d_2(n)$ recursively as follows,

$$r_1^e(n)=r_1(n)(1-p_{d1}(n)), \quad (4)$$

$$r_1^d(n)=r_1(n)-r_1^e(n). \quad (5)$$

$$r'^e_2(n)=r'_2(n)(1-p_{d1}(n)), \quad (6)$$

$$r'^d_2(n)=r'_2(n)-r'^e_2(n). \quad (7)$$

Obviously $\upsilon_1(n)+\lambda_1^e(n)+r_1^e(n)+r'^e_2(n)$ give the total arriving messages which enter the queue of Server 1 at current time slot n. Adding $q_1(n)$ and deducting $\mu_1(n)$ would generate a new queue size of Server 1 at the next time slot $n+1$, i.e.,

$$q_1(n+1)=[\min\{q_1(n)+\lambda_1^e(n)+r_1^e(n)+r'^e_2(n)+\upsilon_1(n)-\mu_1(n), B_1\}]^+ \quad (8)$$

The queue size should be not more than the buffer size B_1.

Like describing the queuing dynamics of Server 1, we can obtain $\lambda_2^e(n)$, $\lambda_2^d(n)$, $r_2^e(n)$ and $r_2^d(n)$ as follows,

$$p_{d2}=[(\lambda_2(n)+r_2(n)+q_2(n)-B_2-\mu_2(n))/(\lambda_2(n)+r_2(n))]^+, \quad (9)$$

$$\lambda_2^e(n)=\lambda_2(n)(1-p_{d2}(n)), \quad (10)$$

$$\lambda_2^d(n)=\lambda_2(n)-\lambda_2^e(n), \quad (11)$$

$$r_2^e(n)=r_2(n)(1-p_{d2}(n)), \quad (12)$$

$$r_2^d(n)=r_2(n)-r_2^e(n). \quad (13)$$

where p_{d2} denotes the drop probability of Server 2 due to the buffer overflow.

Then we can find the queue size of Server 2 at next time slot $n+1$ based on the information at the current time slot n, i.e.,

$$q_2(n+1)=[\min\{q_2(n)+\lambda_2^e(n)+r_2^e(n)-\mu_2(n), B_2\}]^+. \quad (14)$$

The equation for the queue size at Server 1 is different from Server 2 because Server 1 has to receive response messages from Server 2 and retransmit request messages to Server 2 if timer expires, while Server 2 does not need to do so because it is an arbitrary UA (or end-server).

It should be noted that the above equations are useful only when we know how to obtain the arrival retransmitted messages $r_1(n)$ and the response messages $\upsilon_1(n)$ of Server 1, as well as the arrival original messages $\lambda_2(n)$ and the arrival retransmitted messages $r_2(n)$ of Server 2. All of them depend on the queuing and departure processes of Server 1 and Server 2. Therefore they are all intertwined. Our approach is to calculate all quantities in a recursive way and use a divide-and-conquer strategy to solve the complex queuing and departure process as demonstrated in the following subsections.

4.1. Derivation of retransmission process

In order to calculate $r_1(n)$, we first have to look at $r'_1(n)$, the total retransmitted messages generated by all upstream servers of Server 1 at current time slot n. Following the divide-and-conquer strategy, we divide $r'_1(n)$ into 6 components where each component can be calculated easier. The original request messages which arrived at Server 1 at time $n-T_1$ will be retransmitted the first time by upstream server at time n if they have not been processed by Server 1 by time n. These retransmitted messages constitute the first component. Similarly a message arrived at time $n-T_j$, $T_j=(2^j-1)T_1$, will be retransmitted the j^{th} time at time n if the original request messages have not been processed by time n. Let $r'_{1j}(n)$ denote all the j^{th} retransmissions generated by all upstream servers at time n for the original request messages arriving at time $n-T_j$, and $1\le j\le 6$, because there are maximum 6 retransmissions for every original request message [1]. We can obtain the total retransmitted messages $r'_1(n)$ generated by all the upstream servers at current time slot n as

$$r_1'(n)=\sum_{j=1}^{6} r_{1j}'(n)\,. \tag{15}$$

To focus our study on the tandem server, Overload Assumption (e) indicates that the upstream servers have infinite capacity and can process the retransmitted messages without any delay, i.e., $r_1(n)=r'_1(n)$ and $r_{1j}(n)=r'_{1j}(n)$. Then we have

$$r_1(n)=\sum_{j=1}^{6} r_{1j}(n)\,. \tag{16}$$

So our focus now is how to find $r_{1j}(n)$. For the purpose of a clear analysis, we suggest an enqueuing priority scheme to differentiate different types of message arrivals within the same time slot n. The order of the message priority is: the original request messages > the retransmitted request messages from the upstream server > the retransmitted request messages for the downstream server (as shown in Figure 3). The order of the retransmitted message priority is: first-time retransmitted messages > second-time retransmitted messages > … > sixth-time retransmitted messages (as shown in Figure 4). Figure 4 illustrates the enqueuing order of different types of retransmission messages when the queue is not full. The messages with higher priority enter the tail of the queue prior to the messages with lower priority. However, such enqueuing priority scheme does not apply to the traffic arrivals at different time slots, which will follow first-come-first-in policy consistent with the FCFS principle. Therefore, the impact of this priority scheme within a timeslot can be made arbitrarily small when the interval of the time slot decreases.

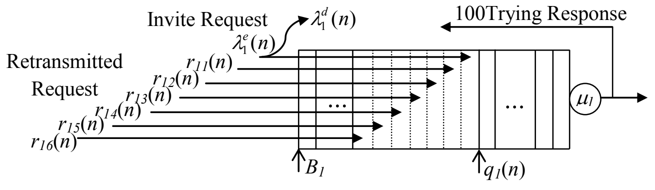


Figure 4. Priority order of the retransmission messages $r_{1j}(n)$.

In order to determine whether an original message needs to be retransmitted by an upstream server, according to SIP RFC [1], we need to know whether the message is still in queue at Server 1 at current time or it has been dropped by Server 1 due to buffer overflow: (a) If the original messages enter the queue at Server 1, future retransmissions of these messages depend on whether these messages are still in queue at their designated retransmission times; (b) If the original messages are dropped, their first retransmissions will happen at their designated first retransmission time for sure. To make things complex is the fact that their second retransmissions depend on whether their first retransmission messages enter the queue or not. If their first retransmission messages enter the queue, future second retransmissions of these messages will again depend on whether their first retransmission messages are still in queue at their designated second retransmission times. If, on the other hand, the first retransmissions are dropped again, the second retransmissions of these messages will be deemed to happen at their designated second retransmission times. This logic will continue until the maximum six retransmissions have been exhausted.

In order to calculate the retransmissions $r_{1j}(n)$ accurately, we propose an innovative strategy to classify various retransmission scenarios as discussed above, as illustrated in Figure 5. Figure 5 provides the key for analyzing the complicated message dropping. For example, $r_{111}(n)$ denotes the first retransmissions at current time n caused by the original messages which entered the queue at time $n-T_1$, and $r_{112}(n)$ denotes the first retransmissions at current time n caused by the original messages which got dropped at time $n-T_1$. The sum of $r_{111}(n)$ and $r_{112}(n)$ generates the total first retransmitted messages $r_{11}(n)$. Similarly $r_{1j}(n)$ consists of $r_{1ji}(n)$ ($i=1,\dots,j+1$, $1<j\le 6$): $r_{1j1}(n)$ denotes the j^{th} retransmissions at current time n caused by the original messages which entered the queue at time $n-T_j$; $r_{1ji}(n)$ ($1<i<j+1$) denote the amount of messages that are being retransmitted the j^{th} time at time n while their corresponding original messages and the related retransmissions had been dropped until their $(i-1)^{\text{st}}$ retransmissions which entered the queue at time $n-T_j+T_{i-1}$; $r_{1jj+1}(n)$ denote the amount of messages that are being retransmitted j^{th}-time at time n while their corresponding original messages and all earlier retransmissions have been dropped. It can be seen that all the scenarios form a tree structure, where each internal node has either one or two children corresponding to the different scenarios we mentioned earlier.

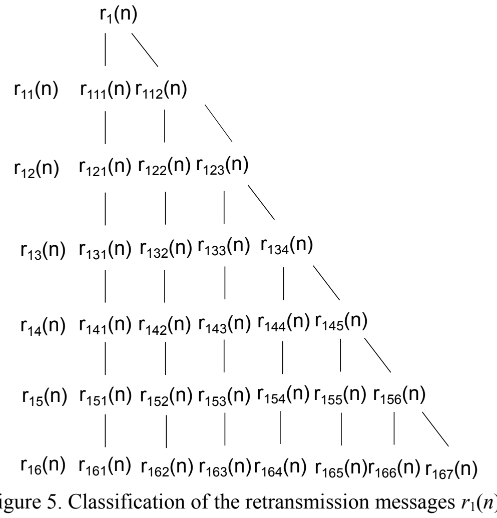

$r_1(n)$
$r_{11}(n)$ $r_{111}(n)$ $r_{112}(n)$
$r_{12}(n)$ $r_{121}(n)$ $r_{122}(n)$ $r_{123}(n)$
$r_{13}(n)$ $r_{131}(n)$ $r_{132}(n)$ $r_{133}(n)$ $r_{134}(n)$
$r_{14}(n)$ $r_{141}(n)$ $r_{142}(n)$ $r_{143}(n)$ $r_{144}(n)$ $r_{145}(n)$
$r_{15}(n)$ $r_{151}(n)$ $r_{152}(n)$ $r_{153}(n)$ $r_{154}(n)$ $r_{155}(n)$ $r_{156}(n)$
$r_{16}(n)$ $r_{161}(n)$ $r_{162}(n)$ $r_{163}(n)$ $r_{164}(n)$ $r_{165}(n)$ $r_{166}(n)$ $r_{167}(n)$

Figure 5. Classification of the retransmission messages $r_1(n)$.

From the above discussion and Figure 5, we can obtain a generalized formula for the total j^{th} retransmission messages $r_{1j}(n)$ by deriving its component $r_{1ji}(n)$ through a recursive approach. To simplify our presentation, we denote T_{ji} to be $T_{ji}=T_j-T_i$ ($1\le i<j\le 6$). Since the response messages enter the queue head and are processed without any delay, we denote

the total processed request messages from the time m to n as $s_1(m,n)=\sum_{l=m}^{n}[\mu_1(l)-\upsilon_1(l)]$. Due to page limit, we omit the detailed mathematical derivation, but provide the recursive equations for obtaining $r_{1j}(n)$ as follows.

$$r_{1j}(n)=\sum_{i=1}^{j+1} r_{1ji}(n)\,, \tag{17}$$

$$r_{1j1}(n)=\min\{[\lambda_1^e(n-T_j)+q_1(n-T_j)-s_1(n-T_j,n)]^+,\ \lambda_1^e(n-T_j)\} \tag{18}$$

$$r_{1ji+1}(n)=\min\{[r^e_{1ii+1}(n-T_{ji})+\sum_{l=1}^{i} r^e_{1il}(n-T_{ji})+\sum_{l=1}^{i-1} r^e_{1l}(n-T_{ji}) + \lambda_1^e(n-T_{ji})+q_1(n-T_{ji})-s_1(n-T_{ji},n)]^+,\ r^e_{1ii+1}(n-T_{ji})\}, \tag{19}$$

$$r_{1jj+1}(n)=r_{1j-1j}(n-T_{jj-1})-r^e_{1j-1j}(n-T_{jj-1})\,, \tag{20}$$

$$r^e_{1jj+1}(n)=r_{1jj+1}(n)(1-p_{d1}(n))\,. \tag{21}$$

We can use the same strategy to obtain the total generated messages $r'_2(n)$ for retransmission created by Server 1 for Server 2. The only difference is that we need to consider the queuing delays and message drops at both Server 1 and Server 2 for non-redundant retransmitted messages r'_{2jj+1}.

4.2. Derivation of departure process

Since the retransmitted messages $r_2(n)$ and the response messages $\upsilon_1(n)$ at Server 1 depends on the original messages $\lambda_2(n)$ at Server 2, hence we need to derive $\lambda_2(n)$.

Calculating $\lambda_2(n)$ is a more formidable task than calculating retransmission rate $r_1(n)$ due to the fact that there may be redundant INVITE requests in the buffer of Server 1. When Server 1 processes an INVITE request, it needs to know whether the message is seen by the server the first time or not. If it is seen the first time, the request will be forwarded to Server 2; otherwise, it will be dropped as a redundant message. To make departure process more complex to calculate is the fact that the queuing delay between a message arrival time and its departure time from Server 1 are also random. So even we know which messages actually enter the queue the first time, we still need to know when they are leaving the server. In the following, we will present our innovative solution in two steps: In the first step, we will identify which messages are seen by Server 1 the first time; in the second step, we will calculate the impact of the delay caused by Server 1. As you will see, the first step turns out to be easy by utilizing the results we developed in the earlier subsection. The second step requires more sophisticated solution.

In addition to the original messages which enter the queue of Server 1, the retransmitted messages which enter the queue of Server 1 the first time are treated as non-redundant messages and forwarded to Server 2. That is, all the non-redundant messages consist of $\lambda_1^e(n)$ and $r_{1jj+1}{}^e(n)$ (j=1,…,6). Thus the arrival rate $\lambda_2(n)$ of Server 2 in current time slot n consists of the departing non-redundant messages from Server 1. Assume $\lambda_{2k}(n)$ (k=1,…,7) represent the departing non-redundant messages which enter the queue as $\lambda_1^e(n)$, $r_{112}{}^e(n)$, $r_{123}{}^e(n)$, $r_{134}{}^e(n)$, $r_{145}{}^e(n)$, $r_{156}{}^e(n)$ and $r_{167}{}^e(n)$ respectively. Our solution next is to derive the departure process for each $\lambda_{2k}(n)$ (k=1,…,7) separately.

Assume the whole SIP network starts running at time n=0. Since varying delays exist between the arrival and departure times for the original messages, the relationship between the departure original messages $\lambda_{2k}(n)$ (k=1,…,7) at current time slot n and all the non-redundant messages $\lambda_1^e(n-d)$ and $r_{1jj+1}{}^e(n-d)$ (j=1,…,6) at previous time slot $n-d$ (d=0,1,2,…,n) is very complex and difficult to determine. The non-redundant messages, which arrived time slot $n-d$ (d=0,1,2,…,n), may or may not contribute to $\lambda_{2k}(n)$ depending on whether they are still in queue at time n.

We propose an innovative approach to obtain $\lambda_{2k}(n)$ based on divide-and-conquer strategy by dividing $\lambda_{2k}(n)$ into individual components so that each of them can be calculated easier. We start with our calculation of $\lambda_{21}(n)$ which represents the departing original messages $\lambda_1^e(n)$.

We denote the amount of original messages $\lambda_1^e(n-d)$ which arrived at time $n-d$ and happened to leave at time n as $\lambda_{21d}(n)$. We solve the challenge posed by the uncertainty that all original messages arriving prior to and at time n may or may not contribute to $\lambda_{21}(n)$ by examining $\lambda_{21d}(n)$ individually. It is easy to see

$$\lambda_{21}(n)=\sum_{d=0}^{n}\lambda_{21d}(n)\,. \tag{22}$$

Clearly some of these $\lambda_{21d}(n)$ will be null.

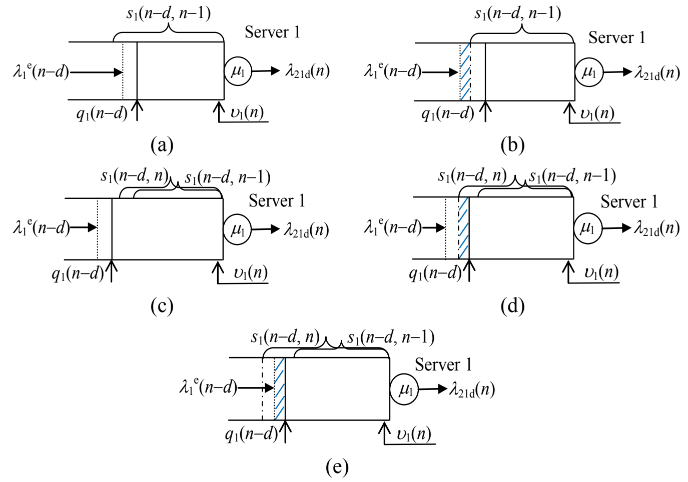


Figure 6. Arrival rate $\lambda_{21d}(n)$ of Server 2 in five different categories.

The enqueuing priority scheme indicates that $\lambda_1^e(n-d)$ will enter the queue and be queued right after $q_1(n-d)$. We consider all possible scenarios that may happen in the following categories:

1. $s_1(n-d,n-1)\geq\lambda_1^e(n-d)+q_1(n-d)$ as indicated in Figure 6(a). This means that the original messages arrived at time $n-d$ have been fully served by the time $n-1$. Therefore we have $\lambda_{21d}(n)$=0 and $[\lambda_1^e(n-d)+q_1(n-d)-s_1(n-d,n-1)]^+$=0.
2. $\lambda_1^e(n-d)+q_1(n-d)>s_1(n-d,n-1)\geq q_1(n-d)$ as indicated in Figure 6(b). This means that the original messages arrived at time n-d have been served partially at time n-1. The remaining messages to be served at time n should be $\lambda_1^e(n-d)+q_1(n-d)-s_1(n-d,n-1)$. We have $\lambda_{21d}(n)=\min\{\lambda_1^e(n-d)+q_1(n-d)-s_1(n-d,n-1),\ \mu_1(n)-\upsilon_1(n)\}$.
3. $s_1(n-d,n-1)\leq q_1(n-d)$ and $s_1(n-d,n)\leq q_1(n-d)$ as indicated in Figure 6(c). This means that none of the original

messages arrived at time n-d has been served by the time n. Therefore we have $\lambda_{21d}(n)=0$ and $[s_1(n-d,n)-q_1(n-d)]^+=0$.

4. $s_1(n-d,n-1)\leq q_1(n-d)$ and $\lambda_1^e(n-d)+q_1(n-d)>s_1(n-d,n)\geq q_1(n-d)$ as indicated in Figure 6(d). This means that, at time n, the server serves the remaining messages in the queue $q_1(n$-$d)$ and then starts serving the original messages arrived at time n-d using the left capacity. We have $\lambda_{21d}(n)=s_1(n-d,n)-q_1(n-d)$.

5. $s_1(n-d,n-1)\leq q_1(n-d)$ and $s_1(n-d,n)\geq q_1(n-d)+\lambda_1^e(n-d)$ as indicated in Figure 6(e). This means that, at the time n, the server starts serving the original messages arrived at time n-d and can finish serving all these messages. We have $\lambda_{21d}(n)=\lambda_1^e(n-d)$.

The above 5 categories are mutual exclusive and have covered all the possible scenarios. By summarizing them together, we can obtain $\lambda_{21}(n)$ through Equation (22) with

$$\lambda_{21d}(n)=[\min\{\lambda_1^e(n-d)+q_1(n-d)-s_1(n-d,n-1),\ \mu_1(n)-\upsilon_1(n),\ s_1(n-d,n)-q_1(n-d),\ \lambda_1^e(n-d)\}]^+,\ d=1,\ldots,n, \tag{23a}$$

When d=0, the original messages arrive at current time slot n. Whether these messages can be served immediately depends on the available capacity $[\mu_1(n)$-$q_1(n)]^+$. Then we have

$$\lambda_{210}(n)=[\min\{\mu_1(n)-\upsilon_1(n)-q_1(n),\ \lambda_1^e(n)\}]^+. \tag{23b}$$

Similar to the derivation of the departing rate for the original messages $\lambda_1^e(n-d)$, we can obtain, the departing rates $\lambda_{2k}(n)$ $(2\leq k\leq 7)$ for the non-redundant retransmitted messages $r_{1k-1k}^e(n)$ which enter the queue for the 1st-time, as

$$\lambda_{2k}(n)=\sum_{d=0}^{n}\lambda_{2kd}(n) \quad k=2,\cdots,7 \tag{24}$$

The sum of the departing messages $\lambda_{2k}(n)$ $(1\leq k\leq 7)$ become the arrival rate of Server 2 in the current time slot n as

$$\lambda_2(n)=\sum_{k=1}^{7}\lambda_{2k}(n)\,. \tag{25}$$

We can use the same strategy to obtain the retransmission messages $r_2(n)$ which are equal to the departure retransmission messages $r'^e_2(n)$, and the response messages $\upsilon_1(n)$ which corresponds to the departure messages $\lambda_2(n)$ and $r_2(n)$.

4.3. Generalization of tandem server to arbitrary topology

Our tandem server topology is quite general except that it does not consider splitting the output of Server 1 to multiple downstream servers and merging the traffic from multiple upstream servers at Server 2.

With the departure process calculated in Section 4.2, it is quite easy to split the output of Server 1 if the splitting process is given based on any splitting policy.

Merging at Server 2 can be treated similarly as the merging at Server 1 except the responses must be sent to their corresponding upstream servers.

We assumed that the upstream servers of Server 1 have infinite capacity. If any upstream server of Server 1 has finite capacity, it can be modelled using similar equations as those for Server 1.

In summary, we can see that Server 1 and Server 2 in our tandem server can be generalized to represent an arbitrary proxy server and an arbitrary UA respectively, two basic components to build an arbitrary SIP network. Our analytical approach can be easily applied to the modeling of an arbitrary SIP network with minor changes. An analytical model for an arbitrary network is very important to conduct a fluid-based simulation for performance evaluation of a large scale network, when an event-driven simulation is infeasible due to expensive computation cost [40].

5. Overload control algorithm

Through the analysis in the Section 4, we have demonstrated the impact of retransmissions. If an original INVITE request is dropped due to buffer overflow or message corruption, retransmission of the request message by the upstream server is necessary. However those retransmissions caused by queuing delay in a server are redundant. They actually make the overloading situation even worse. Reducing this kind of retransmissions will make the retransmission mechanism more effective and increase server utilization. This observation leads us to propose an overload control mechanism that targets at reducing redundant retransmissions.

The key in our proposal is to differentiate necessary retransmissions from redundant ones. We can use either direct or indirect approaches. In indirect approaches, we can estimate whether a downstream server is overloaded based on the way response messages are being received. For example, redundant message ratio can be calculated based on the response messages. This ratio can be used as an indicator about whether a downstream server is overloaded or not. Unfortunately this kind of indirect approaches can cause overreaction under certain special situations.

In this paper, we propose to use a direct approach to get the information about downstream servers. Instead of letting upstream servers guess what are happening at downstream servers, we propose to let downstream servers inform upstream servers explicitly its server utilization dynamically. Server utilization is a good metric to indicate the level of server load. It is easy to implement and available in nearly all existing servers over the market. We propose the downstream servers piggyback this information in response messages. After receiving this message, the upstream server can regulate its retransmission rate based on overload level at downstream servers. To maintain fairness among different users, the retransmission rate can be regulated by randomly suppressing certain portion of retransmissions. In such way, the retransmission probability of each message is related to the load levels at downstream servers.

Without loss of generality, we consider a case that overload happens at Server 1. There are two preset thresholds: low threshold and high threshold. When the average CPU utilization γ_{1avg} exceeds the low threshold, an overload is anticipated at Server 1, thus its upstream server retransmits the messages with a certain probability p_1. We choose a linear function to determine the relationship between p_1 and γ_{1avg}, The linear function not only makes the calculation for p_1 cost-effective, but also achieves satisfactory efficiency of the overload control (as demonstrated by performance evaluation later on). Other nonlinear functions can also be used to determine the relationship between p_1 and γ_{1avg}.

The retransmission probability p_1 is initialized to 1. When the utilization of the downstream server increases, p_1 will decrease until it reaches 0 once the utilization reaches a high threshold, where no message will be retransmitted. If the utilization of downstream servers stays at 1 for a long period to indicate a persistent overload, other existing solutions such as load balancing or call rejections can be used to further mitigate overload. Summary of our overload control mechanism is shown in Figure 7.

Like existing pushback overload control solutions (e.g., SIP overload control RFC [30]), implementation of our overload control algorithm requires minor modification to SIP protocol. We need to define and create a field in the response message to carry the average CPU utilization of the downstream server in order to deliver the overload status to its upstream servers.

Overload Control Algorithm

```
When each retransmission timer fires or expires
  Calculate retransmission probability $p_1$:
    if $\gamma_{1avg} < \gamma_{1low}$
      $p_1 \leftarrow 1$
    else
      if $\gamma_{1low} \le \gamma_{1avg} \le \gamma_{1high}$
        $p_1 \leftarrow (\gamma_{1high}-\gamma_{1avg})/(\gamma_{1high}-\gamma_{1low})$
      else
        $p_1 \leftarrow 0$
```

Varying parameter:

γ_{1avg}: Average CPU utilization of Server 1

Fixed parameters:

γ_{1low} : Low threshold for γ_1

γ_{1high}: High threshold for γ_1

Figure 7. Overload control algorithm based on average CPU utilization.

We have developed an innovative approach to set up the analytical model for a tandem server and demonstrated how to extend our analytical approach to model an arbitrary network. When our overload control algorithm is implemented in every sending server of a SIP network, we need to create a respective analytical model for the fluid-based simulation. The model can help the researchers to speed up the performance evaluation of various SIP overload control solutions using the fluid-based simulation, when a SIP network is scaled up.

Without loss of generality, we start our analysis at Server 1. The total message service rate μ_1 is bounded by the service capacity C_1, i.e., $\mu_1 \le C_1$. At current time slot n, the instantaneous CPU utilization γ_1 can be obtained as

$$\gamma_1(n)=\mu_1(n)/C_1(n). \quad (26)$$

Since stochastic process of both total message service rate μ_1 and service capacity C_1 may cause transient fluctuation in the instantaneous CPU utilization γ_1, we use an exponential weighted moving average filter to calculate the average CPU utilization γ_{1avg} as

$$\gamma_{1avg}(n)=(1-w_\gamma)\gamma_{1avg}(n-1)+w_\gamma\gamma_1(n). \quad (27)$$

where w_γ is the filter weight, and the average CPU utilization γ_{1avg} can be initialized as 0.5. According to the overload control algorithm described by Figure 7, the retransmission probability p_1 generated by the upstream server of Server 1 can be computed as

$$p_1(n)=\min\{[(\gamma_{1high}-\gamma_{1avg}(n))/(\gamma_{1high}-\gamma_{1low})]^+, 1\}. \quad (28)$$

By integrating the retransmission probability p_1 into the theoretical retransmission rate r_1, we can get the actual retransmission rate as r_1p_1. Therefore, we can reuse the analytical model developed for the regular server in Section 4 by replacing the theoretical retransmission rate with the actual retransmission rate. For the recursive equations (i.e., Equations (17) to (21)) to obtain the retransmission rate r_1 of Server 1, we only need to update Equation (21) as

$$r^e_{1jj+1}(n)=r_{1jj+1}(n)(1-p_{d1}(n))p_1(n). \quad (29)$$

6. Performance evaluation and simulation

We evaluate the performance of an overloaded SIP tandem server by performing fluid-based Matlab simulation using the analytical model we have derived. In the mean time, we perform event-driven OPNET simulation to investigate the accuracy of our analytical model for the tandem server. In the OPNET simulator, messages were handled one by one instead of being aggregated over a time slot as in our Matlab simulation. All the sending servers maintained a list of all outstanding messages for tracking retransmissions. Our simulations are based on the SIP network topology depicts by Figure 2. Four user agent clients generated original messages with equal mean rate, and then sent them to a tandem server.

For OPNET simulation, the aggregated mean original message generation rate is equal to the aggregated mean original arrival rate in the corresponding Matlab scenario· Currently there is no measurement result for the workload characteristics in real SIP networks. Our model can simulate any demand arrival process or server service process. Similar to the experiment in [7], in our Matlab and OPNET simulations, arrival rate follows Poisson distribution while the service time of each type of message follows exponential distribution, which has been widely adopted by most researchers [6, 21]. The mean service rates of Server 1 and Server 2 are also set to be the same under normal situations as the mean service rates of the corresponding servers in Matlab simulation. The processing speeds of the upstream servers of Server 1 are set to be so large that their processing times are negligible. A large number of replications need to be simulated to ensure 95% confidence interval. We have run 10 simulation replications for both Matlab simulation and OPNET simulation, and then calculated 95% confidence interval (CI) as $\overline{X} \pm 1.96*\sigma/\sqrt{N}$, where $\overline{X}$ and σ are the mean value and the standard deviation of N=10 replications. We have also run up to 60 replications. The results are very similar except the confidence intervals are much smaller. Therefore 10 replications are enough for illustration purpose. In the simulation plots, $\text{Matlab}_{\text{mean}}$ denotes the mean value of Matlab simulation, while $\text{OPNET}_{\text{mean}}$ and OPNET_{cI} denote the mean value and confidential interval of OPNET simulation respectively.

To demonstrate the effectiveness of our overload control solution, two typical overload scenarios were simulated: (1) Overload at Server 2 due to a server slowdown; (2) Overload at Server 1 due to a demand burst. In each scenario, we performed our simulations without overload control algorithm and with overload control algorithm separately.

The maximum and minimum thresholds for average CPU utilization were set as $\gamma_{1max}=\gamma_{2max}=0.9$ and $\gamma_{1min}=\gamma_{2min}=0.6$ respectively. The moving average filter weight w_γ was 0.1. The buffer sizes of Server 1 and Server 2 were set as B_1=1000 messages and B_2=500 messages respectively. The interval of each time slot was t_s=50 ms and the 1st-time retransmission timer is T_1=500ms [1]. Each scenario was simulated 60s. Since processing a response message took much less time than processing a request message, the proportional ratio was set as 0.5. The mean service capacities of Server 1 and Server 2 were measured based on the processing time of request message, e.g., C_1=1000 request messages/sec indicated that the mean processing times for a request message and a response message were 1ms and 0.5ms respectively. The total message service rate μ was bounded by the service capacity C at each server, i.e., $\mu \leq C$.

6.1. Overload at Server ***2***

In this scenario, we let an initial overload happen at Server 2 due to a server slow down. The mean original message generation rate for each user agent client was 50 messages/sec, i.e., the aggregated original message arrival rate of four user agent clients was λ_1=200 messages/sec. The mean server capacities of the two proxy servers were C_1=1000 messages/sec from time t=0s to t=60s, C_2=100 messages/sec from time t=0s to t=20s (emulating a server slow down due to the routine maintenance), and C_2=1000 messages/sec from time t=20s to t=60s (emulating the normal service).

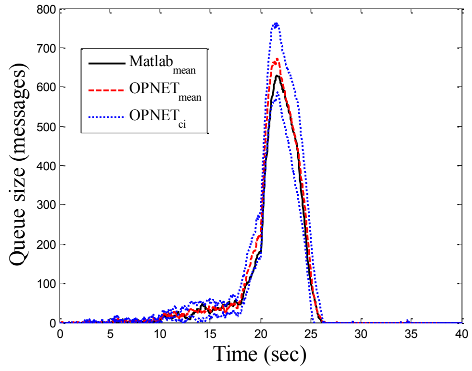


Figure 8. Mean queue size q_1 (messages) of Server 1 and 95% confidential interval versus time upon an initial overload at Server 2 when overload control algorithm was not activated.

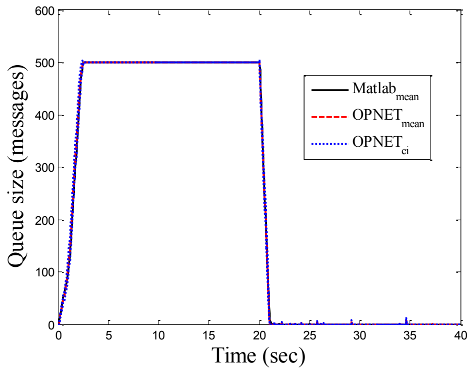


Figure 9. Mean queue size q_2 (messages) of Server 2 and 95% confidential interval versus time upon an initial overload at Server 2 when overload control algorithm was not activated.

1) Performance Without Overload Control Algorithm Applied: Figures 8 to 11 show the dynamic behaviour of a tandem server when overloaded happened at the downstream Server 2. Without overload control algorithm applied, Server 2 became overloaded first, which was followed by a later overload at Server 1. After Server 2 resumed its normal service at time t=20s, Server 1 and Server 2 had the same service capacity. Because Server 1 had to process part of r_1 which would not enter Server 2, the total arrival rate at Server 2 was less than its service capacity. Eventually the overload was cancelled at both Server 1 and Server 2. The confidence intervals in Figures 8 to 11 are quite tight compared to their sample means. Also note the sample means of our analytical model and OPNET model overlap nearly perfectly which indicate a good matching between the two types of models.

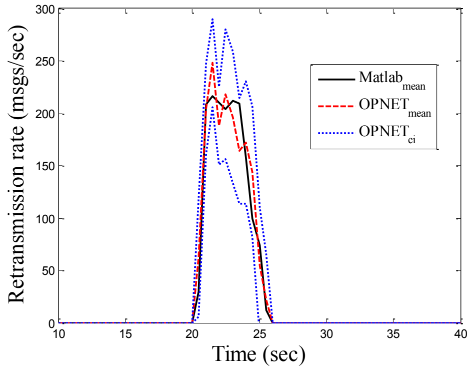


Figure 10. Mean retransmission rate r_1 (messages/sec) of Server 1 and 95% confidential interval versus time upon an initial overload at Server 2 when overload control algorithm was not activated.

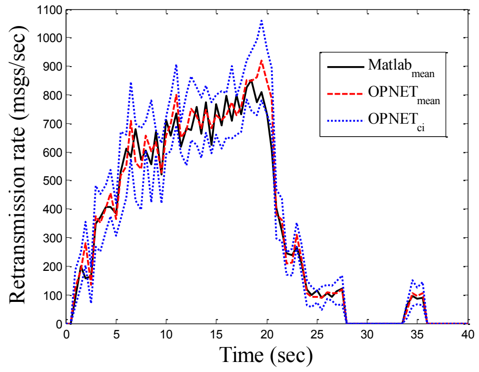


Figure 11. Mean retransmission rate r'_2 (messages/sec) of Server 2 and 95% confidential interval versus time upon an initial overload at Server 2 when overload control algorithm was not activated.

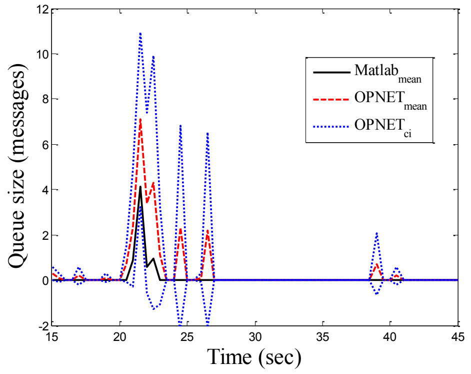


Figure 12. Queue size q_1 (messages) of Server 1 versus time upon an initial overload at Server 2 when overload control algorithm was activated.

2) Effectiveness of Overload Control Algorithm: We activated the overload control algorithm at every sending server. When a server slowdown caused the CPU to perform at full utilization at Server 2 (as shown in Figure 14), our overload control algorithm forbade the retransmissions r'_2 to consume the resources of both Server 1 and Server 2 (as shown in Figure 15). One can see that Server 1 remained almost buffer empty (as shown in Figure 12), and the overload was not propagated from Server 2 to Server 1. After Server 2 resumed its normal service at time t=20s, its buffer became empty very quickly (see Figure 13), and the retransmissions were allowed to recover the original messages dropped by Server 2 (see Figure 15).

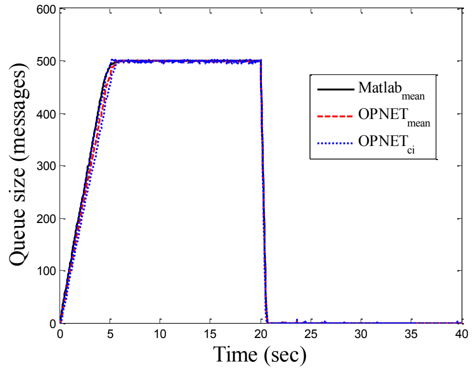


Figure 13. Queue size q_2 (messages) of Server 2 versus time upon an initial overload at Server 2 when overload control algorithm was activated.

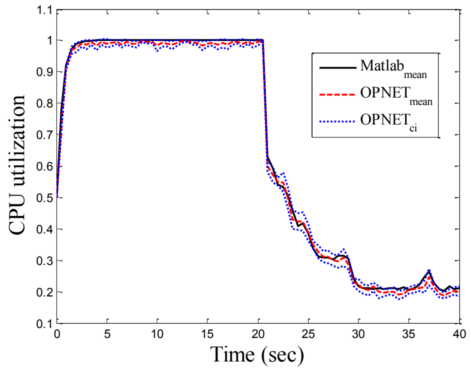


Figure 14. Average CPU utilization γ_{2avg} of Server 2 versus time upon an initial overload at Server 2 when overload control algorithm was activated.

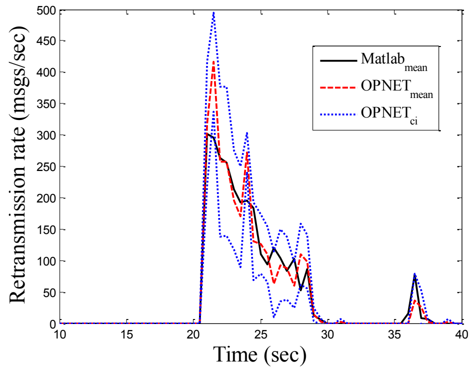
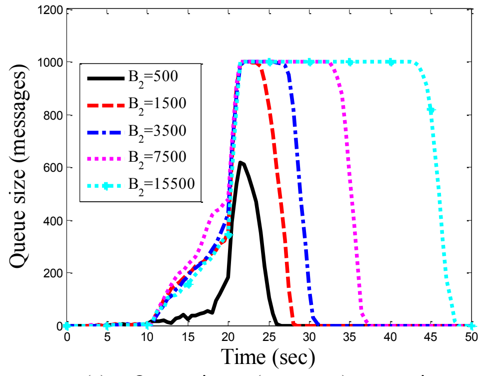


Figure 15. Retransmission rate r'_2 (messages/sec) for Server 2 versus time upon an initial overload at Server 2 when overload control algorithm was activated.

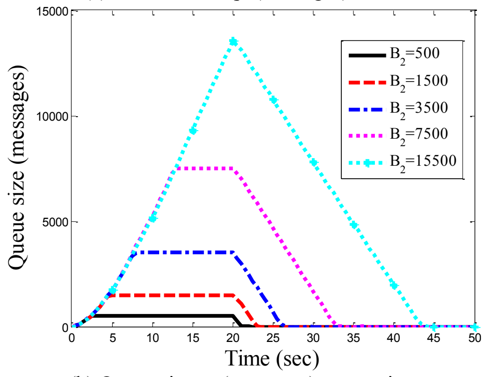
3) Impact of Different Buffer Size: In order to investigate the impact of different buffer sizes on the server performance when an initial overload happened at Server 2 and the overload control algorithm was not activated, we run the fluid-based simulation again.

We would like to observe the server performance under the four different sub-scenarios which includes both extremely small and extremely large buffer sizes: (I) small buffer B_1 and small buffer B_2; (II) large buffer B_1 and small buffer B_2; (III) small buffer B_1 and large buffer B_2; (IV) large buffer B_1 and large buffer B_2.

Since the maximum queue size q_1 of Server 1 was less than 1000 messages when Server 2 had a small buffer size (see Figure 8), increasing the buffer size of Server 1 would not influence the server performance, i.e., sub-scenario II would exhibit the same behaviour as sub-scenario I. Therefore, four sub-scenarios could be merged into two sub-scenarios which our simulations were based on: (I) small buffer B_1=1000 messages and varying buffer B_2; (II) varying buffer B_1 and large buffer B_2=15,500 messages.



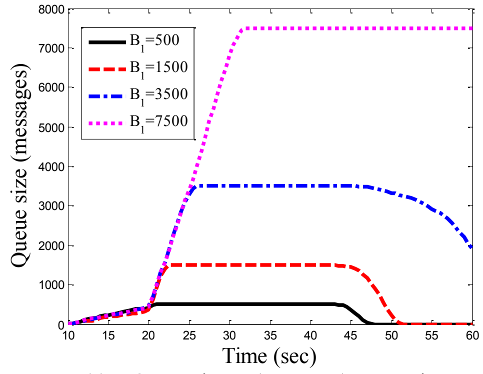
(a) Queue size q_1 (messages) versus time.



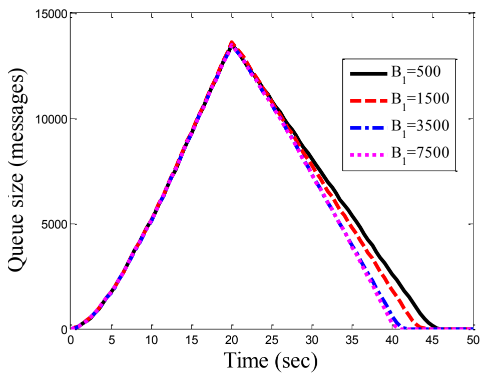
(b) Queue size q_2 (messages) versus time.

Figure 16. Queuing dynamics of a tandem server upon an initial overload at Server 2 when Server 1 had small buffer B_1 (B_1=1000 messages) and Server 2 had different buffer sizes B_2 (messages).

Figures 16 and 17 show the queuing dynamics of both Server 1 and Server 2 upon an initial overload happened at Server 2, when Server 1 and Server 2 had different buffer sizes. The queuing delay is equal to the queue size divided by the server capacity approximately, i.e., $\tau_{q1} \approx q_1/C_1$ and $\tau_{q2} \approx q_2/C_2$. A large queue size corresponds to a long queuing delay.



(a) Queue size q_1 (messages) versus time.



(b) Queue size q_2 (messages) versus time.

Figure 17. Queuing dynamics of a tandem server upon an initial overload at Server 2 when Server 1 had different buffer sizes B_1 (messages) and Server 2 had large buffer B_2 (B_2=15,500 messages).

One can see that the overload was always propagated from Server 2 to Server 1 when Server 1 had a limited service capacity. After Server 2 resumed its normal service at time t=20s, since both Server 1 and Server 2 had the same service capacities, the overload at Server 2 was cancelled eventually no matter what the buffer size of Server 1 or Server 2 was (see Figures 16(b) and 17(b)). Smaller buffer size would help Server 1 to cancel the overload more quickly by rejecting calls, while an extremely large buffer size (e.g., B_1=7500 messages) would maintain the overload at Server 1 continuously, thus bringing down Server 1 eventually (see Figure 17(a)).

4) Comparison of Simulation Time: In order to obtain the simulation result for stochastic systems, we must run a large number of replications to evaluate the system performance. In addition, the original message rates and server capacities may be extremely large in a real SIP network. It would be necessary to compare the total simulation time between fluid-based Matlab simulation and event-driven OPNET simulation.

When both original message arrival rate λ_1 and server service capacities C_1 and C_2 are scaled up 10 and 100 times respectively, e.g., the mean server capacity C_1 becomes 10,000 messages/sec and 100,000 messages/sec respectively. We perform the simulation and record the simulation time for one replication when an initial overload happened at Server 2 and the overload control algorithm was not activated.

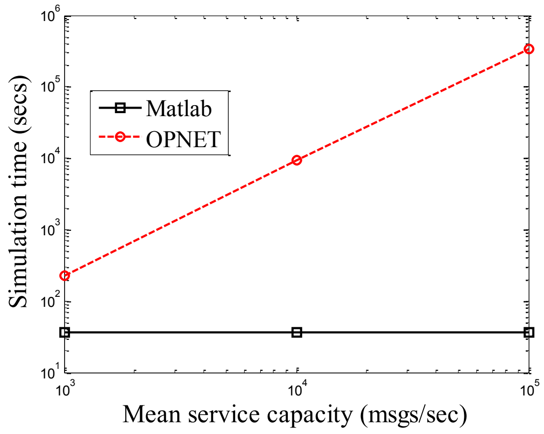


Figure 18. Simulation time (seconds) of different mean server capacities: OPNET simulation vs. Matlab simulation.

Figure 18 shows the simulation time of different server capacities for Matlab simulation and OPNET simulation. Since messages arriving within the same time slot are aggregated and processed together, the computation cost for Matlab simulation is invariant with respect to the server capacity, while the simulation time of OPNET simulation increased exponentially. For example, evaluating the performance of a server with a mean server capacity of 10,000 messages/sec, OPNET simulation took almost 8 days, while Matlab simulation reduced the simulation time 19,000 times to 37 seconds, as shown in Figure 18.

*6.2. Overload at Server **1***

In this scenario, a short period of demand burst overloaded Server 1 from time t=0s to t=30s, emulating a short surge of user demands; Normal original request messages arrived at the overloaded server with a mean rate λ=200 messages/sec, emulating regular user demands. The mean service capacities of Server 1 and Server 2 were C_1=1000 messages/sec and C_2=1000 messages/sec respectively.

1) Performance Without Overload Control Algorithm Applied: Figures 19 and 20 show the dynamic behaviour of a tandem SIP server upon an initial overload at Server 1. One can see that the mean values of Matlab simulation replications stayed inside the confidence interval of OPNET simulation replications, which confirmed the relative accuracy of our analytical model for a tandem server again.

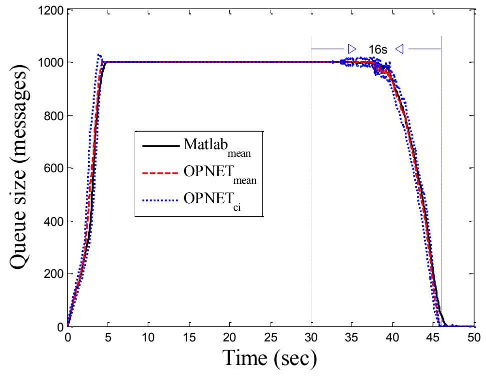


Figure 19. Queue size q_1 (messages) of Server 1 versus time upon an initial overload at Server 1 when overload control algorithm was not activated.

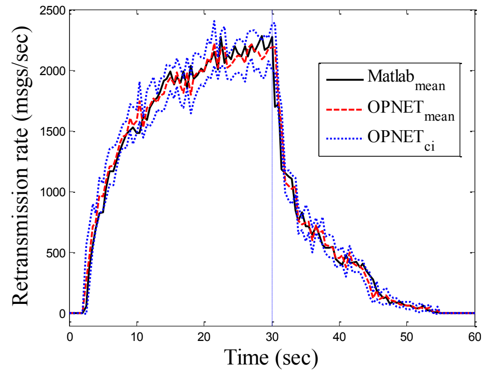


Figure 20. Retransmission rate r_1 (messages/sec) for Server 1 versus time upon an initial overload at Server 1 when overload control algorithm was not activated.

Without overload control applied, the redundant retransmissions were triggered to enter Server 1 (see Figure 20), thus the queue size of Server 1 increased quickly and took around 3.8s to reach the buffer limit (see Figure 19). The finite buffer could mitigate the overload by dropping the messages. At time t=30s, the mean original request rate decreased from

800 messages/sec to 200 messages/sec at Server 1, making the total arrival rate of new original requests and corresponding responses less than the server capacity. Then Server 1 took about 16s to cancel the overload, as marked in Figure 19. Since the service capacities of Server 1 and Server 2 were the same, Server 2 maintained almost empty buffer, and its queue size was not shown here to save the space.

2) Effectiveness of Overload Control Algorithm: Between time t=0s and t=30s, the overload occurred due to a short surge in demands. The overload control algorithm prevented the retransmissions from exacerbating the overload at Server 1, i.e., no retransmissions happened (see Figure 22), thus the queue size of Server 1 increased slowly and took around 8.5s to reach the buffer limit (see Figure 21). After the original message rate decreased at time t=30s, Server 1 became buffer empty and the overload was cancelled within 3s, as marked in Figure 21.

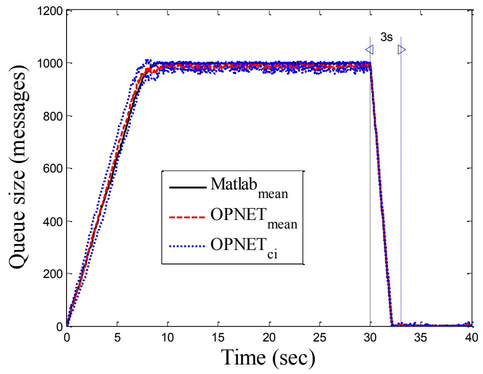


Figure 21. Queue size q_1 (messages) of Server 1 versus time upon an initial overload at Server 1 when overload control algorithm was activated.

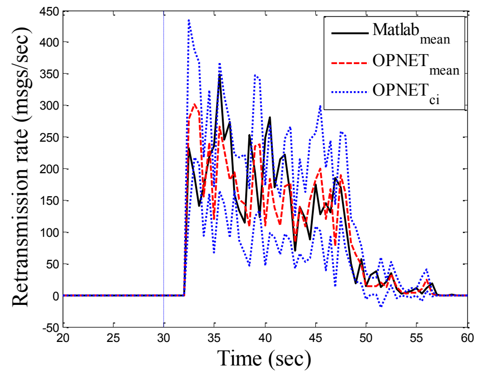


Figure 22. Retransmission rate r_1 (messages/sec) for Server 1 versus time upon an initial overload at Server 1 when overload control algorithm was activated.

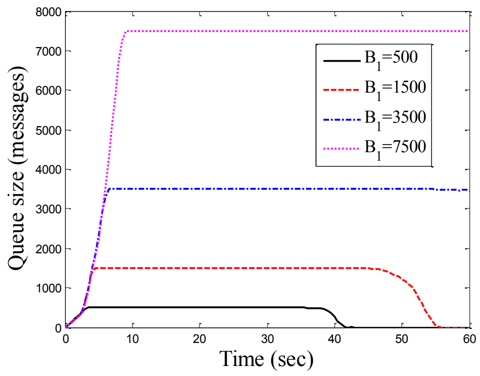


Figure 23. Queue size q_1 (messages) of Server 1 with different buffer size B_1 (messages) versus time upon an initial overload at Server 1 when overload control algorithm was not activated.

3) Impact of Different Buffer Size: In order to investigate the impact of different buffer sizes on the server performance when an initial overload happened at Server 1 and the overload control algorithm was not activated, we run the fluid-based simulation again.

Since the overload at Server 1 would not propagate to the downstream Server 2, we only need to evaluate the queuing dynamics of Server 1 with different buffer sizes, as shown by Figure 23. After the original message arrival rate was decreased significantly at time t=30s, smaller buffer sizes (e.g., 500 messages and 1500 messages) could help Server 1 to cancel the overload within the simulation period, while Server 1 with larger buffer sizes (e.g., 3500 messages and 7500 messages) continued to maintain the overload throughout the whole simulation period.

4) Performance Comparison with Pushback Solution: In addition to server utilization, queuing delay has also been used as a metric to identify the overload state. Queuing-delay controlled pushback solution suggested by [20, 21] has been adopted by SIP overload control RFC [30] recently.

The basic idea of queuing-delay controlled pushback solution [20, 21] is that each overloaded downstream server (e.g., Server 1) advertises a desirable message sending rate to its upstream server. The advertised message sending rate η_1 is calculated as,

$$\eta_1=\mu_1(1-(\tau_{q1}-\tau_{q1o})/t_s), \qquad (30)$$

where μ_1 is the estimated service rate, τ_{q1} is the estimated server queuing delay, τ_{q1o} is the desirable target server queuing delay, t_s is the sampling interval for performing calculation (i.e., the interval of a time slot for our analytical model), and η_1 includes both original and retransmitted request rate (i.e., $\eta_1=\lambda_1+r_1$) [20, 21]. In our simulation, the target server queuing delay was set as τ_{q1o}=400ms, less than 500ms (the default timer for triggering the 1st retransmission [1]).

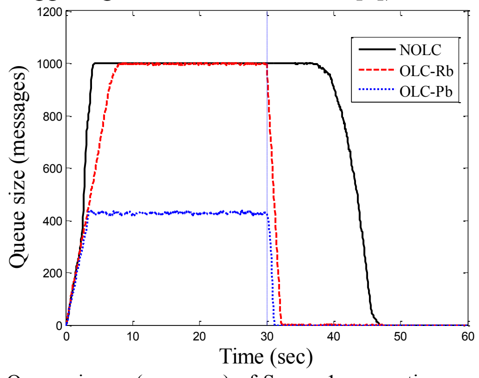


Figure 24. Queue size q_1 (messages) of Server 1 versus time upon an initial overload at Server 1.

By performing the fluid-based Matlab simulation using our analytical model, we use Figures 24 and 25 to depict the queuing dynamics and call rejecting rate of the overloaded Server 1 under non-overload control algorithm, our retransmission-based algorithm, and queuing delay controlled pushback algorithm suggested by [20, 21]. In the two plots, NOLC denotes no overload control algorithm was activated; OLC–Rb denotes retransmission-based overload control algorithm was activated; and OLC−Pb denotes queuing-delay controlled pushback overload control algorithm was activated.

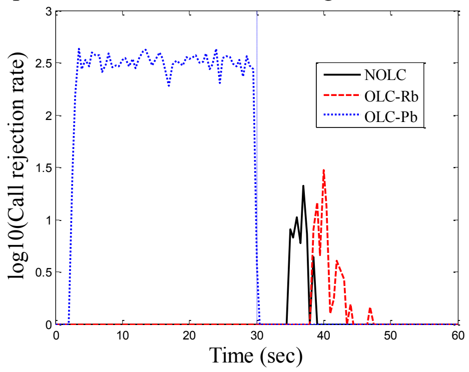


Figure 25. Call rejecting rate (calls/sec in logarithm scale) of Server 1 versus time upon an initial overload at Server 1.

As shown in Figure 24, after a short surge in demands ended at time *t*=30s, pushback solution cancelled the overload using about 1 second, and retransmission-based solution spent 3 seconds (also shown in Figure 21). Without any overload control solution applied, the server had to wait about 16s to become buffer empty (also shown in Figure 19).

However, the cost of quickly cancelling the overload is not free. For a short-term overload that lasted for 30 seconds, queuing-delay controlled pushback solution rejected 10 times more calls than both retransmission-based solution and the case without overload control solution; the call rejection rate of retransmission-based solution was comparable with the case without overload control solution, as shown in Figure 25. The reason behind the fact that the retransmission-based solution also had to reject a small amount of calls unintentionally is the retransmission-based solution forbids the necessary retransmissions for recovering the original INVITE requests dropped by the overloaded Server 1 due to the buffer overflow.

One can see that compared with the pushback solution, our retransmission-based solution achieves a better trade-off between the speed to cancel the overload and the call rejection rate when the overload only lasts a short period.

6.2. Remarks

A small buffer size can help a server to cancel the overload by dropping excessive original messages, which may block more calls and cause unnecessary revenue loss.

1) Short-term Overload: If an overload lasts a short period, our overload control algorithm can mitigate the overload effectively and prevent network collapse from overload propagation by restricting retransmission rate, while maintaining the original request rate to avoid blocking excessive calls. After the overloaded server performs its normal operation, the overload is cancelled rapidly, and the upstream servers can generate the retransmission freely to retain full reliability.

2) Long-term Overload: If an overload lasts a long period, existing pushback solutions need to be activated and the original request rate should be reduced and some calls have to be blocked intentionally. Since pushback solutions mitigate the overload by rejecting calls intentionally and causing the revenue loss, they cancel the overload much faster than the proposed retransmission-based solution in this paper.

To achieve a satisfactory trade-off between call blocking probability and the time to cancel the overload, we suggest integrating our retransmission-based solution with other pushback solution. During the real-time implementation, each server specifies a threshold for the overload period, and creates an overload timer to monitor the overload status: (1) activates the retransmission-based solution to control retransmissions only, at the initial stage of an overload; (2) activates the pushback solutions to block some calls intentionally, if the overload period exceeds its threshold. A good value for the threshold of the overload period can be chosen as the total time allowed for maximum 6 retransmissions during hop-by-hop INVITE transaction [1], which is approximately 30s.

3) Non-INVITE Transactions: As each server needs to consume CPU to process non-INVITE transactions, different type of messages pose potential threat to the overload collapse. Our overload control algorithm is applicable for mitigating the overload caused by non-Invite transactions. Our modeling approach can be naturally extended to include non-INVITE messages for non-INVITE transactions by introducing a proportional ratio to take into account the different processing times for INVITE and non-INVITE messages.

7. Conclusions

In order to propose an effective overload control mechanism, we have studied the impact of retransmission on the overload by modeling the dynamic behaviour of a SIP network where each server has finite buffer. We have introduced our novel modeling strategy by modeling a tandem server, and then demonstrated how to extend our innovative methodology to model an arbitrary SIP network. Unlike various existing signaling models based on Poisson distributed arrival rate and exponentially distributed service time, our study considered a general case that both arrival rate and service rate for signaling messages are arbitrary stochastic processes. Our three key analytical results are: (1) the formulation of different types of message drops; (2) the formulation of different types of retransmission messages due to queuing delay or message drops and (3) the formulation of the departure process through the analysis of all possible departure scenarios. Our solution is computationally efficient, and the scalability of the analytical model allows the network operators or the researchers to evaluate the performance of a large scale SIP network that is well beyond the capabilities of current discrete event simulators.

Our mathematical analysis and simulation have led us to the conclusion that redundant retransmissions can make overload much worse and therefore should be controlled before any other mechanisms are applied.

Based on our modeling results, we have proposed a novel yet simple feedback control algorithm to mitigate the short-term SIP overload effectively by reducing retransmission rate based on downstream server utilization. We have demonstrated how to extend our modeling strategy to model a SIP server with our overload control mechanism. When combined with existing overload control mechanisms, our proposal can improve overload situation and increase server utilization significantly.

The effectiveness of our overload control algorithm and the accuracy of our analytical model have been confirmed by performance evaluation of both Matlab simulation and OPNET simulation. Our study on the impact of buffer size demonstrates that a small buffer size can help a server to cancel the overload by blocking excessive calls. However, this approach will degrade quality of service and reduce revenue for carriers. In addition, for a short-term overload, the retransmission-based solution achieves a better trade-off between the call rejection rate and the speed to cancel the overload when compared with the pushback solution.

In the future work, we would like to investigate how to combine the retransmission-based solution with the pushback solution to achieve a satisfactory performance under different overload scenarios and network topologies.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments. This work was supported by the NSERC grant #CRDPJ 354729-07 and the OCE grant #CA-ST-150764-8.

References

1. Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, Handley M and Schooler E. SIP: Session Initiation Protocol. *IETF RFC 3261*, June 2002.
2. 3rd Generation Partnership Project. http://www.3gpp.org.
3. Faccin SM, Lalwaney P and Patil B. IP Multimedia Services: Analysis of Mobile IP and SIP Interactions in 3G Networks. *IEEE Communications Magazine* 2004, 42(1): 113-120.
4. Rosenberg J. Requirements for Management of Overload in the Session Initiation Protocol. *IETF RFC 5390*, December 2008.
5. Shen C and Schulzrinne H. On TCP-based SIP Server Overload Control. In: *Proceedings of IPTComm*, Munich, Germany, August 2010.
6. Noel E and Johnson CR. Initial simulation results that analyze SIP based VoIP networks under overload. In: *Proceedings of 20th International Teletraffic Congress*, 2007, pp. 54-64.
7. Hilt V and Widjaja I. Controlling Overload in Networks of SIP Servers. In: *Proceedings of IEEE ICNP*, Orlando, Florida, October 2008, pp. 83-93.
8. SIP Express Router. http://www.iptel.org/ser/.
9. Dacosta I, Balasubramaniyan V, Ahamad M and Traynor P. Improving Authentication Performance of Distributed SIP Proxies. In: *Proceedings of IPTComm*, Atlanta, GA, July 2009.
10. Jiang H, Iyengar A, Nahum E, Segmuller W, Tantawi A and Wright C. Load Balancing for SIP Server Clusters. In: *Proceedings of IEEE INFOCOM*, April 2009, pp. 2286-2294.
11. Warabino T, Kishi Y and Yokota H. Session Control Cooperating Core and Overlay Networks for “Minimum Core Architecture. In: *Proceedings of IEEE Globecom*, Honolulu, Hawaii, December 2009.
12. Huang L. Locating Interested Subsets of Peers for P2PSIP. In: *Proceedings of International Conference on New Trends in Information and Service Science*, 2009, pp. 1408-1413.
13 Xu L, Huang C, Yan J and Drwiega T. De-Registration Based S-CSCF Load Balancing in IMS Core Network. In: *Proceedings of IEEE ICC*, Dresden, Germany, 2009.
14. Geng F, Wang J, Zhao L and Wang G. A SIP Message Overload Transfer Scheme. In: *Proceedings of ChinaCom*, October 2006.
15. Kitatsuji Y, Noishiki Y, Itou M and Yokota H. Service Initiation Procedure with On-demand UE Registration for Scalable IMS Services. In: *Proceedings of The Fifth International Conference on Mobile Computing and Ubiquitous Networking*, Seattle, WA, April 2010.
16. Ohta M. Overload protection in a SIP signaling network. in *Proceedings of International Conference on Internet Surveillance and Protection*, 2006.
17. Garroppo RG, Giordano S, Spagna S and Niccolini S. Queueing Strategies for Local Overload Control in SIP Server. In: *Proceedings of IEEE Globecom*, Honolulu, Hawaii, 2009.
18. Amooee AM and Falahati A. Overcoming Overload in IMS by Employment of Multiserver Nodes and Priority Queues. In: *Proceedings of International Conference on Signal Processing Systems*, May 2009, pp. 348-352.
19. Dacosta I and Traynor P. Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks. In: *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 2010.
20 Noel E and Johnson CR. Novel Overload Controls for SIP Networks. In: *Proceedings of 21st International Teletraffic Congress*, 2009.
21. Shen C, Schulzrinne H and Nahum E. SIP Server Overload Control: Design and Evaluation. In: *Proceedings of IPTComm*, Heidelberg, Germany, July 2008.
22 Abdelal A and Matragi W. Signal-Based Overload Control for SIP Servers. In: *Proceedings of IEEE CCNC*, Las Vegas, NV, January 2010.
23. Ohta M. Overload Control in a SIP Signaling Network. In: *Proceeding of World Academy of Science, Engineering and Technology*, Vienna, Austria, March 2006, pp. 205-210.
24. Montagna S and Pignolo M. Comparison between two approaches to overload control in a Real Server: “local” or “hybrid” solutions? In: *Proceedings of IEEE MELECON*, April 2010, pp. 845-849.
25. Wang YG. SIP Overload Control: A Backpressure-based Approach. *ACM SIGCOMM Computer Communications Review* 2010, 40(4): 399-400.
26. Homayouni M, Jahanbakhsh M, Azhari V and Akbari A. Overload control in SIP servers: Evaluation and improvement. In: *Proceedings of IEEE ICT*, Doha, Qatar, April 2010, pp. 666-672.
27. Yang J, Huang F and Gou SZ. An Optimized Algorithm for Overload Control of SIP Signaling Network. In: *Proceedings of 5th International Conference on Wireless Communications, Networking and Mobile Computing*, September 2009.
28. Sun J, Tian R, Hu J and Yang B. Rate-based SIP Flow Management for SLA Satisfaction. In: *Proceedings of IEEE/IFIP IM*, New York, USA, June 2009, pp. 125-128.
29. Hong Y, Huang C and Yan J. Analysis of SIP Retransmission Probability Using a Markov-Modulated Poisson Process Model. In: *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, Osaka, Japan, April 2010, pp. 179–186.
30. Gurbani VK, Hilt V and Schulzrinne H. Session Initiation Protocol (SIP) Overload Control. *IETF Internet-Draft*, draft-ietf-soc-overload-control-07, January 2012.
31. Hilt V, Noel E, Shen C and Abdelal A. Design Considerations for Session Initiation Protocol (SIP) Overload Control. *IETF Internet-Draft*, draft-hilt-soc-overload-design-00, May 2010.
32. Fras M, Mohorko J and Cucej Z. Modeling of captured network traffic by the mimic defragmentation process. *SIMULATION: Transactions of the Society for Modeling and Simulation* 2011, 87(5): 437-448.
33. Klampfer S, Kotnik B, Svecko J, Mezgec Z, Mohorko J and Chowdhury A. MIMO simulator of call server input lines occupancy. *SIMULATION: Transactions of the Society for Modeling and Simulation* 2011, 87(5): 423-436.
34. Nahm EM, Tracey J and Wright CP. Evaluating SIP server performance. In: *Proceedings of ACM SIGMETRICS*, San Diego, CA, US, 2007, pp. 349–350.
35. Yang JP. Stateless Fair Admission Control. *SIMULATION: Transactions of the Society for Modeling and Simulation* 2011, 87(3): 253-261.
36. Hong Y, Huang C and Yan J. Mitigating SIP Overload Using a Control-Theoretic Approach. In: *Proceedings of IEEE Globecom*, Miami, FL, U.S.A, December 2010.
37. Hong Y, Huang C and Yan J. Controlling Retransmission Rate For Mitigating SIP Overload. In: *Proceedings of IEEE ICC*, Kyoto, Japan, June 2011.
38. Hong Y, Huang C and Yan J. Design Of A PI Rate Controller For Mitigating SIP Overload. In: *Proceedings of IEEE ICC*, Kyoto, Japan, June 2011.
39. Hong Y, Huang C and Yan J. Applying Control Theoretic Approach To Mitigate SIP Overload. *Telecommunication Systems*, 2012, in press.
40. Liu Y, Presti FL, Misra V, Towsley DF and Gu Y. Scalable fluid models and simulations for large-scale IP networks. In: *Proceedings of ACM SIGMETRICS*, June 2003, pp. 91-101.
41. Hong Y, Huang C and Yan J. Stability Condition for SIP Retransmission Mechanism: Analysis and Performance Evaluation. In: *Proceedings of IEEE SPECTS*, Ottawa, Canada, July 2010, pp. 387-394.
42. Hong Y, Huang C and Yan J. Modeling Chaotic Behaviour of SIP Retransmission Mechanism. *International Journal of Parallel, Emergent and Distributed Systems*, iFirst, 2012.

43. Govind M, Sundaragopalan S, Binu KS and Saha S. Retransmission in SIP over UDP – Traffic Engineering Issues. In: *Proceedings of International Conference on Communication and Broadband Networking*, Bangalore, India, May 2003.

Yang Hong received B.S. degree in electronic engineering from Shanghai Jiao Tong University, China, M.E. degree in electrical engineering from National University of Singapore, Singapore, and Ph.D. degree in electrical engineering from University of Ottawa, Canada. His research interests include SIP overload control, Internet traffic classification, Internet congestion control, modeling and performance evaluation of computer networks, and industrial process control.

Changcheng Huang received his B. Eng. in 1985 and M. Eng. in 1988 both in Electronic Engineering from Tsinghua University, Beijing, China. He received a Ph.D. degree in Electrical Engineering from Carleton University, Ottawa, Canada in 1997. From 1996 to 1998, he worked for Nortel Networks, Ottawa, Canada where he was a systems engineering specialist. He was a systems engineer and network architect in the Optical Networking Group of Tellabs, Illinois, USA during the period of 1998 to 2000. Since July 2000, he has been with the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada where he is currently an associate professor. Dr. Huang won the CFI new opportunity award for building an optical network laboratory in 2001. He was an associate editor of *IEEE Communications Letters* from 2004 to 2006. Dr. Huang is a senior member of IEEE.

James Yan is currently an adjunct research professor with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. Dr. Yan received his B.A.Sc., M.A.Sc., and Ph.D. degrees in electrical engineering from the University of British Columbia, Vancouver, Canada. He was a telecommunications systems engineering senior manager in Bell-Northern Research (1982-1996) and in Nortel (1996 to 2004). In those positions, he was responsible for projects in performance analysis of networks and products, advanced technology research, planning new network services and architectures, development of network design methods and tools, and new product definition. From 1988 to 1990, he participated in an exchange program with the Canadian Federal Government, where he was project prime for the planning of the evolution of the nationwide federal government telecommunications network. Dr. Yan is a member of IEEE and Professional Engineers Ontario.