Modeling chaotic behaviour of SIP retransmission mechanism

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 3 December 2011; final version received 5 December 2011*)

The SIP retransmission mechanism can cause SIP network collapse with short term overload. In this paper, we investigate with a fluid modeling approach the chaotic behaviour of the SIP retransmission mechanism in SIP networks. We capture the complex correlation structure in SIP systems through a detailed and novel queuing analysis. To dimension a buffer size which can avoid unnecessary message drop in a SIP server, we develop a sufficient condition for a stable SIP system analytically based on our fluid model. We also apply our fluid model to the simulation of a complex SIP system. We compare the simulation results achieved through our fluid models with those based on OPNET® event-driven approach to demonstrate the validity of our approach.

Keywords: SIP; network protocol; queuing model; performance evaluation

1. Introduction

Session Initiation Protocol (SIP) [1] has been widely deployed for significantly growing session-oriented applications in the Internet, such as Internet telephony (or Voice over IP (VoIP)), instant messaging and video conference. As a signaling protocol, SIP is responsible for creating, modifying and terminating session in a mutual real-time communication [2]. With WiMAX and the 3rd Generation 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 [3-5].

Request for Comments (RFC) 5390 [6] 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 [7-23] discussing how to control SIP overload through call rejection or load balancing. However these mechanisms can either increase call rejection rate or cause SIP server underutilized. Carriers do not want to see either of these to happen. We believe that both problems can be avoided in most situations by understanding the root cause of server overloading and widespread SIP network failures.

The goal of this paper is to develop a new analytical fluid 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 SIP hop-by-hop retransmission mechanism, originally designed for reliability purpose, can cause such server failures in a SIP network.

The contributions of this paper are: (1) Proposing an innovative approach for modeling SIP networks with SIP retransmission mechanisms. The arrival and service processes of each SIP server can be arbitrary, which makes our modeling approach quite general. We have successfully analyzed and formulated the complex retransmission and departure processes, i.e., our two major breakthroughs, as will be explained in later sections. (2) Demonstrating the impact of the initial overload condition on a SIP system by both analytical and simulation approaches. Our results indicate that different initial queue sizes may generate totally different behaviour patterns for an overload SIP server (as shown in Section 5.1), which indicates the chaotic nature of SIP retransmission mechanism. This chaotic behaviour is caused by redundant retransmissions triggered by short traffic burst or temporary server slow down. The overload may migrate from a downstream server to its upstream server due to retransmission mechanism (as shown in Section 5.2). A sufficient condition is provided for maintaining system stability; (3) Making performance comparison between fluid-based Matlab simulation and event-driven OPNET simulation in a SIP network to show our fluid model to be quite accurate. Fluid-based simulation can reduce simulation time dramatically, when a large amount of signaling traffic has to be processed in 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 below the triggering condition. Operators can schedule their maintenance periods to avoid server overload conditions. Researchers can develop more efficient overload control algorithms that aim at reducing redundant retransmissions instead of rejecting calls.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 briefly reviews SIP protocol. In Section 4 we apply our modeling approach to a single overloaded server and present the condition for a stable system. We then demonstrate how to extend our modeling approach to a tandem server and an arbitrary network. In Section 5 we demonstrate the chaotic nature of a SIP system under Poisson-distributed demands through simulation. We also make performance comparison between fluid-based simulation and event-driven simulation to claim the accuracy of our fluid model. Conclusions are given in Section 6.

2. Related work

Recent collapses of SIP servers in the real carrier networks (e.g., VoIP outages in British Telecom, Vonage and Wanadoo [24]) started attracting research interests on SIP overload problem. Experimental evaluation of SIP server showed the overload collapse behaviour in [25]. Some attempts have been made to address the SIP overload [22, 26]. For example, three window-based feedback algorithms were proposed to adjust the message sending rate of the upstream SIP servers based on the queue length [11]. Both centralised and distributed SIP overload control mechanisms were studied in [10]. Retry-after control, processor occupancy control, queue delay control and window based control were proposed to improve goodput and prevent overload collapse in [7]. SIP Express Router (SER) developed advanced tuning and routing modules to mitigate overload caused by large subscriber populations or abnormal operational conditions [13]. Request batching was combined with parallel execution to improve call throughput and reduce call failure rate significantly in [15]. A novel authentication protocol was developed to reduce the load on the centralized authentication database dramatically and enhance the overall security of a carrier-scale VoIP network in [16]. The experiments in these works revealed that the retransmission mechanism can exacerbate the overload. However, the impact of the retransmission on the CPU load of a SIP server has never been well studied.

A demand burst or routine server maintenance may create a temporary long queue. Such long initial queue size may continue to stimulate retransmissions and crash the server, even after the server resumes its normal service, and the effective CPU utilization is low. We consider the SIP system exhibiting some chaotic behaviour [27] in the sense that: (1) the system is sensitive to initial conditions, i.e., a small difference in the initial conditions can lead to dramatic differences in system states as time evolves; (2) the dynamics of the system can be described by a set of nonlinear differential equations as shown in Section 4.

To the best of our knowledge, no existing works have investigated the impact of initial overload condition on the SIP system performance. Modeling the retransmission mechanism of SIP can help the researchers fully understand the chaotic behaviour of SIP systems before any effective overload control mechanism can be developed and studied.

In addition, event-driven simulation has been widely used for evaluating network performance. Its computation cost grows linearly with network sizes and message volumes [28]. When event-driven simulation is used to evaluate a SIP network, each outstanding SIP message requires a timer being maintained. When overloads happen, outstanding messages are built up, and simulators need to increase the number of timers dramatically to track message retransmissions. Tracking and manipulating these timers consume large amount of memory and CPU time which make the simulation process extremely slow, thus in some cases, cause simulators to crash and terminate simulations unexpectedly. In order to avoid this kind of simulator crash, the experiments (e.g., [10]) have to take the following actions: (1) reduce original message generation rate and their retransmission timers in a SIP network; (2) limit the buffer size and drop a large portion of demand bursts; (3) limit the processing capacities of SIP servers. These actions have limited the scalability of the simulation approach significantly. Furthermore, most of the interesting events in SIP, e.g., server crash, are considered as highly correlated rare events. For the study of these correlated rare events, extremely long simulated time and large number of replications are typically required to get any meaningful statistical result.

Therefore, it is necessary to develop some other approaches to simplify the CPU-consuming timer-tracking process. Fluid-based models have been successfully used to achieve scalability by aggregating events into time slots [29, 30]. By running a fluid-based simulation on a desktop PC, Liu et al. evaluated Transmission Control Protocol (TCP) performance of a large scale IP network consisting of hundreds of routers and thousands of high bandwidth links supporting millions of flows, which is unachievable by current event-driven simulators [29]. In addition, Liu et al. performed both fluid-based simulation and event-driven simulation on a small-size network in [29]. Their simulation results demonstrated that fluid-based simulation can achieve comparable performance with event-driven simulation if the sampling interval is small enough. With acceptably small errors, fluid-based simulation has attained significant speedups when compared with event-driven simulation [29, 30].

Fluid models for Asynchronous Transfer Mode (ATM) and data networks have been created and analyzed widely. The book [31] provides a comprehensive review on these models. An analytical model was proposed to evaluate a finite-buffered multistage interconnection network (MIN) whose communication structure can be Ethernet switch or ATM switch [32]. However, these models all deal with bandwidth-constrained networks, while SIP networks are server (CPU) constrained. None of the existing models have considered retransmissions. Even the TCP models in [29, 31] do not consider retransmissions. It is difficult to capture retransmissions with fluid model because retransmissions generate duplicate messages and create a complex correlation structure due to various timers.

We introduced the fluid model for SIP systems with finite buffer in [33] where our focus was to study the impacts of message loss. A major conclusion in [33] was that small buffers can mitigate overload with a cost of rejecting a large number of calls. Our purpose here is to investigate the chaotic behaviour of SIP systems and find the condition to maintain a stable system. Therefore in this paper we let buffer sizes be infinite. The stable condition was briefly introduced in [34] initially. Both papers [33] and [34] assumed that the processing time for response messages was negligible when compared with the processing time for the request messages. However, in some cases, this assumption can cause major errors. For example, a large number of response messages were generated when the downstream Server 2 just resumed its normal service, as indicated by Fig. 15 in Section 5.2 later on. The total workload generated by this response message burst is not negligible even though the workload generated by each response message is much smaller than the processing time of the request message. Other than the treatment of the response messages, the paper [34] also has some technical errors in its mathematical proof. This paper will significantly improve those results through new analysis and simulation on tandem server with infinite buffer, accurate treatment of response messages, robust mathematical proof, a new lemma on buffer dimensioning, OPNET simulation results, and vigorous illustration of chaotic behaviour.

We will show how we solve the problem of modeling the retransmission mechanism in SIP with infinite buffer by using a fluid model in Section 4. This is the major enabler of the contributions we make in this paper.

3. SIP protocol overview

A SIP network mainly consists of the two basic entities: user agent (UA) and proxy server. A user agent can act as a client (UAC) or as a server (UAS). Fig. 1 depicts a typical procedure of a session establishment. To set up a call, a UAC sends an "Invite" request to a UAS via the two proxy servers. The proxy server or the UAS returns a provisional "100Trying" response to confirm the receipt of the "Invite" request. The UAS returns an

"180Ring" response after confirming that the parameters are appropriate. It also evicts a "200OK" message to answer the call. The UAC sends an "ACK" response to the UAS after receiving the "200OK" message. Finally the call session is established and the media communication is created between the UAC and the UAS through the SIP session. The "Bye" request is generated to finish 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" message. The call will then be rejected.

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.

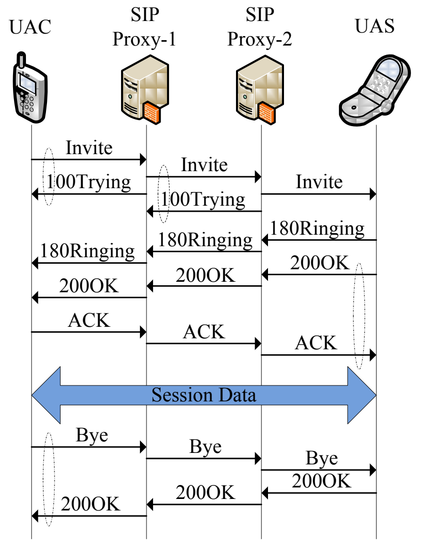


Figure 1. A typical procedure of session establishment.

To provide reliability for SIP Invite messages, SIP protocol incorporates a hop-by-hop retransmission mechanism [1, 35]. For each hop, the sender starts the first retransmission of an original message at T_1 seconds and the time interval doubles after every retransmission (exponential back-off), if the corresponding 100Trying response 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]. However, each SIP sender cannot detect whether retransmissions are stimulated by actual message loss or excessive time delay due to the overload, and it always retransmits the messages after corresponding timers expire.

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 [1]. However, it has been demonstrated that TCP cannot prevent SIP overload collapse [10, 17]. Therefore nearly all vendors choose to run SIP over User Datagram Protocol (UDP) instead of TCP.

In a typical SIP system, numerous SIP servers are connected into a network. It would be necessary to investigate the interaction between the neighbouring servers and know how an overloaded server can cause a whole SIP network to break down. The topology of a real SIP network can be quite complex, it is hard to select a specific large network to represent an arbitrary network. However there are in general two types of SIP nodes as shown in Fig. 1: user agents vs. proxy servers. In some cases, application servers can take the place of user agents. In this paper, we start with a single server which represents the most overloaded proxy server in a SIP network. We then demonstrate how to extend the results for single server to tandem server scenario where one represents a proxy server and the other represents a UA. In Section 4.3, we discuss how to generalize our results to an arbitrary SIP network.

Our approach to be developed in this paper can be applied to any queuing and scheduling policy. SIP RFC [1] does not specify any specific queuing and scheduling policy, which is left to vendors to choose for their competitive advantage. Without loss of generality, we choose a First-In-First-Out (FIFO) based queuing policy as suggested by a vendor. This policy has the benefit of easy to implement in real system and is therefore the first choice for many vendors. The queuing policy and its associated SIP system are defined below:

(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 [11]. 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. A smaller time slot corresponds to more accurate simulation result, but a longer simulation time. However it becomes meaningless to reduce time slot to the level of a message processing time. The default interval of a time slot is chosen as 50ms in this paper, because it can provide a desirable accuracy for the simulation scenarios we select. According to the SIP RFC [1], the corresponding first retransmission time T_1 would be 0.5s and 10 timeslots equivalently. We use t and n to denote time and timeslot respectively;

(c) Queuing-priority Mechanism: The SIP RFC [1] does not specify the queuing and scheduling discipline to be deployed by a SIP server. We let a SIP server maintain a First-In-First-Out (FIFO) queue for messages arriving at different time-slots. All request messages enter the tail of a server queue. The FIFO queue model reflects the common practice by most vendors today [10, 13]. For messages which arrive within the same time slot, we choose the order of the enqueuing priority to be: the original request messages > the retransmitted request messages from the upstream servers > the retransmitted request messages for the downstream servers (as shown in Figures 2 and 4). It is easy to see that this special priority mechanism has negligible impact on average total arrival rate if the interval of the time slot is very small. There is no enqueuing difference for the messages arriving at different time slots;

(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 let 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, but cannot be neglected for some overload scenarios;

(e) Overload Assumption: When overload happens in the network, at any time, one of the servers will be the most overloaded one among all the overloaded servers. It becomes the bottleneck server. In our single overloaded server scenario, we will focus on this bottleneck server by assuming all its upstream and downstream servers have infinite capacities and therefore are not overloaded;

(f) Tandem Choice: Our tandem server model consists of Server 1 and Server 2, where Server 1 represents an arbitrary proxy server, while Server 2 represents an arbitrary user agent server (as shown in Fig. 4). Server 1 may have multiple upstream servers; The generic nature of the tandem server allows us to generalize our conclusions to an arbitrary SIP network;

(g) Tandem Capacity Setting: To study the interactions among multiple servers, we consider a tandem server scenario where the upstream servers connecting with a tandem server have large capacity enough to process all requests, retransmissions, and response messages immediately without any delay. However, the capacities of the two servers in the tandem server are limited;

(h) Buffer Size Selection: The buffer sizes for both single overloaded server scenario and tandem server scenario are selected to be infinite. This allows us to show the chaotic behaviour that queue size increases forever under certain initial conditions. This will also lead to the conclusion that increasing buffer size will not mitigate overload. Practical buffer sizes vary with the actual service rates and system configuration plans. With the memory becoming cheaper and CPU becoming faster, typical buffer sizes are likely to become larger.

4. Model and analysis

To provide a better understanding of our modeling approach, we create a fluid model for a single overloaded server first. Then we discuss how to extend our modeling approach to a tandem server and an arbitrary network with minor modifications.

4.1. Modeling single overloaded SIP server

In a real SIP network, a single overloaded server may represent the most overloaded one among a group of SIP servers at a specific time. To start with, we assume all the SIP servers other than the overloaded one have infinite capacities (i.e., Overload Assumption (e)). Fig. 2 depicts the queuing dynamics of a single overloaded server. The key challenge to analyzing SIP servers is the various types of messages a SIP server has to process. 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 messages are: original Invite message 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. Response arrival rate $\upsilon_1(n)$ is determined by message departure process $\lambda_2(n)$. Given that both original message arrival process $\lambda_1(n)$ and server service process $\mu_1(n)$ can be arbitrarily distributed, how to obtain retransmission process $r_1(n)$ and departure process $\lambda_2(n)$ is a great challenge for the SIP modeling.

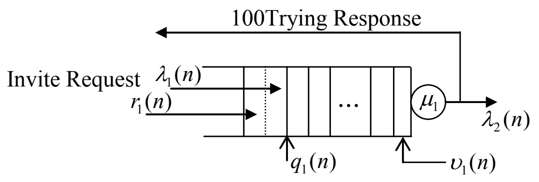


Figure 2. Queuing dynamics of an overloaded SIP server ($\lambda_1(n)$ denotes original message arrivals, $r_1(n)$ denotes retransmitted message arrivals, $\upsilon_1(n)$ denotes response message arrivals, $q_1(n)$ denotes queue size, $\mu_1(n)$ denotes service rate, and $\lambda_2(n)$ denotes original message departures).

It is clear both $r_1(n)$ and $\lambda_2(n)$ depend on the queuing process. Let $q_1(n)$ denote the queue size at time n. We can obtain the queue size $q_1(n+1)$ at next time slot $n+1$ based on the information at current time slot n, i.e.,

$$q_1(n+1)=[q_1(n)+\lambda_1(n)+r_1(n)+\upsilon_1(n)-\mu_1(n)]^+, \quad (1)$$

where $\lambda_1(n)+r_1(n)+\upsilon_1(n)$ give the total arrival messages at current time slot n. Adding $q_1(n)$ and deducting the service rate $\mu_1(n)$ would generate a new queue size $q_1(n+1)$ at next time slot $n+1$, as described by Eq. (1).

Eq. (1) is useful only when we can obtain the arrival retransmitted messages $r_1(n)$ and the response messages $\upsilon_1(n)$, both depend on the queuing and departure processes. 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.

4.1.1. Obtaining retransmitted messages $r_1(n)$

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 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

$$r'_1(n)=\sum_{j=1}^{6} r'_{1j}(n). \quad (2)$$

Since the upstream servers have infinite capacity and can process the retransmitted messages without any delay (i.e., Overload Assumption (e)), we have

$$r_1(n)=r'_1(n). \quad (3)$$

So our focus now is how to find $r'_{1j}(n)$. In order to determine whether an original message needs to be retransmitted by an upstream server, we need to know whether the message is still in queue at the overloaded server at current time. Since the message arrival process and the server service process are arbitrary, the queuing delay for every original message becomes variable. This makes it a complex task.

We now describe our novel method to calculate $r'_{1j}(n)$ by characterizing the original messages that have been serviced for the time-out period T_j and the remaining messages that are still in queue at current time.

At time $n-T_j$, the original message arrivals were $\lambda_1(n-T_j)$ and the queue size was $q_1(n-T_j)$. To decide how many of these messages will be retransmitted again at time n, we need to know how many of them are still in queue at time n. Since the response messages enter the queue head and are processed without any delay, we define the total processed request messages from time m to n as $s_1(m,n)=\sum_{l=m}^{n}[\mu_1(l)-\upsilon_1(l)]$. The overloaded SIP server can process $s_1(n-T_j,n)$ request messages during the past T_j time slots.

After T_j time slots, the remaining request messages of those queued prior to the time slot $n-T_j$ become $[q_1(n-T_j)-s_1(n-T_j,n)]^+$.

According to Queuing-priority Mechanism (c) and Response Treatment (d), the new arrival original messages $\lambda_1(n-T_j)$ entered the queue tail prior to the retransmitted messages $r_1(n-T_j)$. Therefore the retransmission of a request message also depends on the arrival rate $\lambda_1(n-T_j)$, but not on $r_1(n-T_j)$.

Without counting $r_1(n-T_j)$, the remaining request messages in the queue at current time n become $[\lambda_1(n-T_j)+q_1(n-T_j)-s_1(n-T_j, n)]^+$.

This includes the original messages and the response messages which arrived at time $n-T_j$, and the queued messages right before the time slot $n-T_j$. However, only the remaining original arrival messages $\lambda_1(n-T_j)$ need to be retransmitted at time n, so we use minimum function to obtain the j^{th} retransmissions $r'_{1j}(n)$ as

$$r'_{1j}(n)=\min\{[\lambda_1(n-T_j)+q_1(n-T_j)-s_1(n-T_j, n)]^+, \lambda_1(n-T_j)\}. \quad (4)$$

4.1.2. Obtaining response messages $\upsilon_1(n)$

Any downstream server of the overloaded server is required to reply a response message to acknowledge the receipt of a request message sent from the overloaded server. Since the downstream servers can process the incoming messages without any delay (i.e., Overload Assumption (e)) and the propagation delay is negligible (i.e., Delay Consideration (a)), the arrival response messages $\upsilon_1(n+1)$ at next time slot $n+1$ should be equal to the departure original messages $\lambda_2(n)$ at current time n, i.e.,

$$\upsilon_1(n+1)=\lambda_2(n). \quad (5)$$

Therefore the key to calculate the $\upsilon_1(n)$ is to calculate the departure process $\lambda_2(n)$.

Let the whole SIP network start running at time n=0. Since varying queuing delays exist between the arrival and departure times for the original messages, the relationship between the departure original messages $\lambda_2(n)$ at current time slot n and all the arrival original messages $\lambda_1(n-d)$ at previous time slot $n-d$ (d=0,1,2,…,n) is very complex and difficult to determine. The original messages $\lambda_1(n-d)$, which arrived time slot $n-d$ (d=0,1,2,…,n), may or may not contribute to $\lambda_2(n)$ depending on whether they are still in queue at time n. For a causal system, due to queuing and processing delay, only the original messages which arrive at time n or earlier may contribute to $\lambda_2(n)$.

We propose an innovative approach to obtain $\lambda_2(n)$ based on divide-and-conquer strategy by dividing $\lambda_2(n)$ into individual components so that each of them can be calculated easier.

We denote the amount of original messages which arrived at time $n-d$ and happens to leave at time n as $\lambda_{2d}(n)$. 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 $\lambda_2(n)$ by examining $\lambda_{2d}(n)$ individually. It is easy to see

$$\lambda_2(n) = \sum_{d=0}^{n} \lambda_{2d}(n)\ . \qquad (6)$$

Clearly some of these $\lambda_{2d}(n)$ will be null.

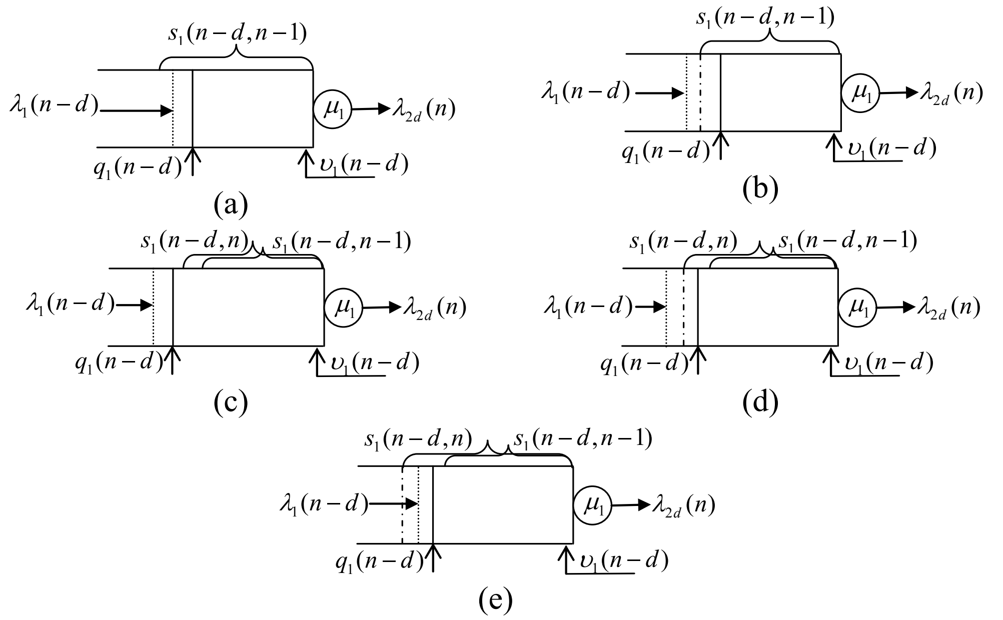


Figure 3. Departure original messages $\lambda_{2d}(n)$ under five different scenarios: $\lambda_1(n-d)$ denotes original message arrivals at time slot $n-d$, $\upsilon_1(n-d)$ denotes response message arrivals at time slot $n-d$, $q_1(n-d)$ denotes queue size at time slot $n-d$, $s_1(n-d,n-1)$ denotes the total processed messages between time slot $n-d$ and time slot $n-1$, and $\lambda_{2d}(n)$ denotes departure original messages which arrived at time slot $n-d$ and leave at current time slot n.

By definition, $\lambda_{2d}(n)$ is related to $\lambda_1(n-d)$ and the queuing process. At time $n-d$ at the overloaded server, the arrival messages from its upstream servers were $\lambda_1(n-d)$ for the original requests and $r_1(n-d)$ for the retransmissions; the response messages from the downstream servers were $\upsilon_1(n-d)$; the service capacity was $\mu_1(n-d)$; the queue size was $q_1(n-d)$. Queuing-priority Mechanism (c) and Response Treatment (d) indicate that $\lambda_1(n-d)$ were queued right after $q_1(n-d)$, while $\upsilon_1(n-d)$ entered the queue head. Fig. 3 describes all five possible scenarios between the processed messages $s_1(n-d,n-1)$ and the queued messages $q_1(n-d)$. Identifying these scenarios is critical for calculating whether there is any departure message from Server 1 at time n.

1. $s_1(n-d,n-1)\geq\lambda_1(n-d)+q_1(n-d)$ as indicated in Fig. 3 (a). This means that the original messages arrived at time $n-d$ have been fully served by the time $n-1$. Therefore we have $\lambda_{2d}(n)=0$ and $[\lambda_1(n-d)+q_1(n-d)-s_1(n-d,n-1)]^+=0$.
2. $\lambda_1(n-d)+q_1(n-d)>s_1(n-d,n-1)\geq q_1(n-d)$ as indicated in Fig. 3 (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 $\lambda_1(n-d)+q_1(n-d)-s_1(n-d,n-1)$. We have $\lambda_{2d}(n)=[\min\{\lambda_1(n-d)+q_1(n-d)-s_1(n-d,n-1), \mu_1(n)-\upsilon_1(n)\}]^+$.
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 Fig. 3 (c). This means that none of the original messages arriving at time $n-d$ has been served by the time n. Therefore we have $\lambda_{2d}(n)=0$ and $[s_1(n-d,n)-q_1(n-d)]^+=0$.
4. $s_1(n-d,n-1)\leq q_1(n-d)$ and
$\lambda_1(n-d)+q_1(n-d)>s_1(n-d,n)\geq q_1(n-d)$ as indicated in Fig. 3 (d). This means that, at time n, the server serves the response messages $\upsilon_1(n-d)$ and the remaining messages in the queue $q_1(n-d)$ and then starts serving the original messages which arrived at time n-d using the left capacity. We have $\lambda_{2d}(n)=s_1(n-d,n)-q_1(n-d)$.

5. $s_1(n-d,n-1) \leq q_1(n-d)$ and $s_1(n-d,n) \geq \lambda_1(n-d)+q_1(n-d)$ as indicated in Fig. 3 (e). This means that, at the time n, the server starts serving the original messages arriving at time $n-d$ and can finish serving all these messages. We have $\lambda_{2d}(n)=\lambda_1(n-d)$.

The above 5 categories are mutual exclusive and have covered all the possible scenarios. By summarizing them together, we can obtain the departure rate $\lambda_{2d}(n)$ of the overloaded server as

$$\lambda_{2d}(n)=[\min\{\lambda_1(n-d)+q_1(n-d)-s_1(n-d, n-1),\ \mu_1(n)-\upsilon_1(n),\ s_1(n-d, n)-q_1(n-d),\ \lambda_1(n-d)\}]^+,\ d=1,\ldots,n, \tag{7a}$$

When d=0, the original messages arrive at current time slot n. Whether these messages can be served immediately depends on the available capacity $[\mu_1(n)-\upsilon_1(n)-q_1(n)]^+$. Then we can obtain $\lambda_{20}(n)$ as

$$\lambda_{20}(n)=[\min\{\mu_1(n)-\upsilon_1(n)-q_1(n),\ \lambda_1(n)\}]^+. \tag{7b}$$

Combining Eqs. (1) to (7) will give us a complete description of the dynamic behaviour of an overloaded SIP server. Due to its chaotic characteristic, performance evaluation in next section shows that this complex, sometimes chaotic pattern can drive the SIP server into instability. The fluid model described by Eqs. (1) to (7) allows us to conduct a simple fluid-based Matlab simulation. This will significantly reduce the simulation time comparing to the traditional event-driven simulation where a large number of timers for retransmissions need to be tracked.

4.1.3. Stability condition of SIP retransmission mechanism

The messages accumulated by a transient overload (e.g., a demand burst or a server slowdown) create an initial queue size when a SIP server returns to its normal service state. Such initial queue size may bring a queuing delay long enough to trigger the retransmissions of old remaining original messages in the queue as well as all the new incoming original messages. It is important for deriving a stability condition that the server can cancel the overload introduced by an initial queue size.

Without loss of generality, we consider the "Invite-Trying" request-response pair with a deterministic arrival rate λ_1, a mean service rate μ_1, an initial queue size $q_1(0)$, and a mean response rate υ_1.

***Theorem* 1**: If the initial queue size $q_1(0)$ created by a demand burst can satisfy a sufficient stability condition described by Eq. (8), then the SIP server is stable and it can cancel the overload.

$$q_1(0) < \min_{1\leq i\leq J}\left\{(2^{J+1}-1)(\mu_1-\upsilon_1)T_1, \frac{(2^{J+1}+3\times 2^i-i-4)(\mu_1-\upsilon_1)T_1-((i-1)2^i+1)\lambda_1 T_1}{i+1}\right\}, \tag{8}$$

where J denotes maximum retransmissions which can be handled by server service capacity effectively, and $J=\lfloor(\mu_1-\upsilon_1-\lambda_1)/\lambda_1\rfloor$.

Proof: To maintain stability in a SIP server, the total average message arrival rate should be less than the average service rate. If there are at most i retransmissions for any original Invite request message, a conservative condition to maintain stability is $((i+1)\lambda_1+\upsilon_1)/\mu_1 \leq 1$, which is equivalent to

$$i \leq (\mu_1-\upsilon_1-\lambda_1)/\lambda_1. \tag{9}$$

To achieve the above sufficient stability condition, we need to guarantee that the original messages from both the initial queue size and the new arrivals are not retransmitted more than j times at any time, where we denote J as $J=\lfloor(\mu_1-\upsilon_1-\lambda_1)/\lambda_1\rfloor$, and $(\mu_1-\upsilon_1)$ represents the available service capacity for the original and retransmitted request messages. According to SIP RFC 3261 [1], J is bounded by six maximum retransmissions, i.e., $J \leq 6$. Then we update the equivalent stability condition in Eq. (9) as

$$i \leq J=\lfloor(\mu_1-\upsilon_1-\lambda_1)/\lambda_1\rfloor. \tag{10}$$

To avoid (J+1) retransmissions for the original messages in the initial queue size, we need to make sure that they have all been processed by the time $T_{J+1}=(2^{J+1}-1)T_1$. This gives us the first condition in Eq. (8):

$$q_1(0)<(\mu_1-\upsilon_1)T_{J+1}. \tag{11}$$

To avoid (J+1) retransmissions for any newly arrival original messages, the server should have finished processing all the queuing request messages $q_1(t)$ by the retransmission time $t+T_{J+1}$ using its available service capacity $(\mu_1-\upsilon_1)$. Thus the queue size at any time t should satisfy

$$q_1(t)<(\mu_1-\upsilon_1)T_{J+1}. \qquad 0 \leq t < \infty \tag{12}$$

The relationship between the condition described by Eq. (8) and the condition described by Eq. (12) is not trivial. From Figures 5 and 8 in performance evaluation section later on, it is clear that the maximum queue size may happen at any time depending on the initial queue size. By examining those two figures, we can see that the entire time can be divided into five different sets. These five different sets are mutual exclusive and have covered all the time periods with different queuing behaviours due to the chaotic nature of the SIP retransmission mechanism.

To guarantee that the condition (12) is satisfied at any time, all we need to do is to guarantee the condition (12) is satisfied in each set. We now examine each set individually.

1. We first consider the queue sizes at each specified retransmission times $T_i=(2^i-1)T_1$ using Eqs. (1) to (4) as follows,

$$q_1(T_i)=q_1(T_{i-1})-2^{i-1}(\mu_1-\upsilon_1-i\lambda_1)T_1+[q_1(0)-(\mu_1-\upsilon_1)T_i]^+. \quad (13)$$

2. We next consider the queue sizes between any two neighbouring retransmission times T_{i-1} and T_i. Eqs. (1) to (4), (10) and (13) lead to

$$q_1(t)=q_1(T_{i-1})-(\mu_1-\upsilon_1-i\lambda_1)(t-T_{i-1})<q_1(T_{i-1}), \quad (14)$$

The inequality in Eq. (14) indicates that the queue size is decreasing continuously with a slope of $(\mu_1-\upsilon_1-i\lambda_1)$ during the time period. However, at time $t=T_i$, the *ith* retransmission for the remaining $[q_1(0)-(\mu_1-\upsilon_1)T_i]$ messages from the initial queue size $q_1(0)$ is triggered, resulting in a sudden increase in the queue size described by (13). Then when $0<t\leq T_J$, the condition described by (12) becomes

$$q_1(T_i)<(\mu_1-\upsilon_1)T_{J+1},\ 1\leq i\leq J. \quad (15)$$

Given the condition of Eq. (11), we consider the worst case with $q_1(0)-(\mu_1-\upsilon_1)T_i\geq 0$. Using recursive substitution for Eq. (13), we can obtain the queue size

$q_1(T_i)=(i+1)q_1(0)-\sum_{k=1}^{i}2^{k-1}(\mu_1-\upsilon_1-k\lambda_1)T_1-\sum_{k=1}^{i}(2^k-1)(\mu_1-\upsilon_1)T_1$, which can be reorganized as

$$q_1(T_i)=(i+1)q_1(0)-\sum_{k=1}^{i}2^{k-1}(\mu_1-\upsilon_1)T_1+\frac{d}{dx}\sum_{k=1}^{i}\lambda_1T_1x^k\bigg|_{x=2}-\sum_{k=1}^{i}2^k(\mu_1-\upsilon_1)T_1+i(\mu_1-\upsilon_1)T_1,$$

We can further simplify the queue size as

$$q_1(T_i)=(i+1)q_1(0)+((i-1)2^i+1)\lambda_1T_1-(3\times 2^i-i-3)(\mu_1-\upsilon_1)T_1. \quad (16)$$

Combining Eqs. (15) and (16), we can obtain the second condition in Eq. (8) as

$$q_1(0)<\frac{(2^{J+1}+3\times 2^i-i-4)(\mu_1-\upsilon_1)T_1-((i-1)2^i+1)\lambda_1T_1}{i+1}, 1\leq i\leq J. \quad (17)$$

3. We then consider the time period $T_J<t<T_{J+1}$. From Eqs. (1) to (4), (10), (12) and (15), we have

$$q_1(t)=q_1(T_J)-(\mu_1-\upsilon_1-(j+1)\lambda_1)(t-T_J)\leq q_1(T_J)<(\mu_1-\upsilon_1)T_{J+1}. \quad (18)$$

This means the queue size is non-increasing during the time period $T_J<t<T_{J+1}$.

4. Next, we consider the retransmission time $t=T_{J+1}$. Since Eq. (11) indicates $[q_1(0)-(\mu_1-\upsilon_1)T_{J+1}]^+=0$, from Eqs. (1) to (4), (10) and (15), we can obtain

$$q_1(T_{J+1})=q_1(T_J)-2^J(\mu_1-\upsilon_1-(j+1)\lambda_1)T_1+[q_1(0)-(\mu_1-\upsilon_1)T_{J+1}]^+\leq q_1(T_J)<(\mu_1-\upsilon_1)T_{J+1}. \quad (19)$$

5. Finally, we consider the time period $t>T_{J+1}$. From Eqs. (1) to (4), (10) and (19), we have

$$q_1(t)=q_1(T_{J+1})-(\mu_1-\upsilon_1-(j+1)\lambda_1)(t-T_{J+1})\leq q_1(T_{J+1})<(\mu_1-\upsilon_1)T_{J+1}. \quad (20)$$

Combining Eqs. (11), (14), (15), (17), (18), (19) and (20), we can reach a sufficient stability condition for the initial queue size described by Eq. (8). We complete the proof. □

In a real SIP network where arrival rate, service rate, and response rate may be arbitrarily distributed, the SIP operators can apply Theorem 1 to determine the stability condition using the mean values of arrival rate, service rate and response rate. A moving average filter can be used to measure the mean values λ_1, μ_1, and υ_1. In addition, from the above proof process, we can see that Equations (11) and (16) provide the minimum finite buffer size required to avoid message loss when the system is stable.

***Lemma* 1**. If a SIP server satisfies the stable condition in Theorem 1, the minimum finite buffer size required to avoid message loss will be

$$\max_{1\leq i\leq J}\left\{q_1(0),\ (i+1)q_1(0)+((i-1)2^i+1)\lambda_1T_1-(3\times 2^i-i-3)(\mu_1-\upsilon_1)T_1\right\}.$$

4.2. Modeling SIP tandem server

We will demonstrate how to apply our novel technique to model a tandem server by making some minor modifications in this subsection. Then discuss modifications required for an arbitrary SIP network in next subsection.

In our tandem server model as shown in Fig. 4, Server 2 is selected to be a UA and Server 1 is an arbitrary server (i.e., Tandem Choice (f)). According to Tandem Capacity Setting (g), both servers 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, and therefore is simpler than the single server model discussed earlier. However, 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. This is different from the single server scenario in Section 4.1. Fortunately our novel technique for analyzing departure process used in Section 4.1.2 can help solve this problem.

Since Server 2 (i.e., the downstream server of Server 1) has limited service capacity, Server 1 may need to retransmit Invite messages $r'_2(n)$ for Server 2 due to the queuing delay of Server 2, and therefore adding a new type of retransmission arrival process $r'_2(n)$ to Server 1 when overload happens at Server 2. It should be noted that $r_2(n)$, the retransmitted messages sent by Server 1 arriving at Server 2, is not equal to $r'_2(n)$ due to delay in Server 1. This is the only difference between Server 1 and the single server we discussed in Section 4.1, as shown in Figures 2 and 4.

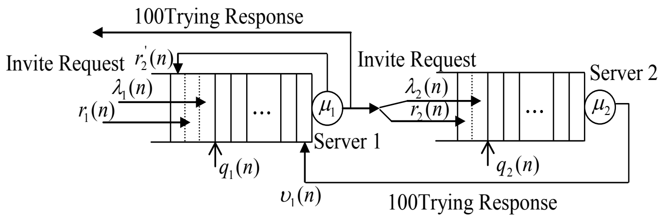


Figure 4. Queuing dynamics of a tandem SIP server (*For Server 1*, $\lambda_1(n)$ denotes original message arrivals, $r_1(n)$ denotes retransmitted message arrivals from upstream server, $r'_2(n)$ denotes retransmitted messages created for Server 2, $q_1(n)$ denotes queue size, $\mu_1(n)$ denotes service rate; *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).

In the following parts, we will describe how to get λ_2(n), $r'_2(n)$, and $r_2(n)$. They all depend on the queuing processes at both servers. So we start with the queuing processes.

We consider the downstream Server 2 first. Similar to Eq. (1) in Section 4.1, we can obtain the queue size $q_2(n+1)$ of Server 2 at next time slot $n+1$ based on the information at current time slot n, i.e.,

$$q_2(n+1)=[q_2(n)+\lambda_2(n)+r_2(n)-\mu_2(n)]^+, \tag{21}$$

where at current time slot n at Server 2, $q_2(n)$ denotes the queue size; $\lambda_2(n)$ denotes the arrival original request messages; $r_2(n)$ denotes the arrival retransmitted messages sent by Server 1 corresponding to λ_2(n); $\mu_2(n)$ denotes the processed messages.

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 times out while Server 2 does not need to do so because it does not have a downstream server based on Tandem Choice (f).

Similar to Eq. (1) in Section 4.1, we can have the queuing dynamics for Server 1 as follows,

$$q_1(n+1)=[q_1(n)+\lambda_1(n)+r_1(n)+r'_2(n)+\upsilon_1(n)-\mu_1(n)]^+, \tag{22}$$

where at current time slot n at Server 1, $q_1(n)$ denotes the queue size; $\lambda_1(n)$ denotes the aggregated arrival original request messages from the upstream servers of Server 1; $r_1(n)$ denotes the aggregated retransmitted messages from the upstream servers of Server 1 corresponding to λ_1(n); $\upsilon_1(n)$ denotes the response messages from Server 2 corresponding to $\lambda_2(n)$; $\mu_1(n)$ denotes the processed messages; $r'_2(n)$ denotes the messages generated by Server 1 for the retransmission of original request messages $\lambda_2(n)$ arriving at Server 2.

4.2.1. Obtaining retransmission rate $r_1(n)$

Similar to Eqs. (2), (3), and (4), we have

$$r_1'(n) = \sum_{j=1}^{6} r_{1j}'(n). \tag{23}$$

$$r'_{1j}(n)=\min\{[\lambda_1(n-T_j)+q_1(n-T_j)-s_1(n-T_j, n)]^+, \lambda_1(n-T_j)\}, \tag{24}$$

$$r_1(n)=r'_1(n), \tag{25}$$

where $r'_{1j}(n)$ denotes the j^{th} retransmission for the original request messages arriving at Server 1 at time $n-T_j$ and $T_j=(2^j-1)T_1$.

Remark 1: If the upstream server has finite capacity, Server 1 can represent a generic server in any arbitrary SIP network. Then we can obtain $r_1(n)$ from $r'_1(n)$ using the similar steps for deriving the departure process $\lambda_2(n)$ in Section 4.1.2.

4.2.2. Obtaining arrival rate $\lambda_2(n)$

Due to the limited capacity of Server 1, $\lambda_2(n)$ depends on the departure process of Server 1 as mentioned earlier. Following the similar steps for deriving the departure process in Section 4.1.2, we can obtain the following equations，

$$\lambda_2(n) = \sum_{d=0}^{n} \lambda_{2d}(n), \tag{26}$$

$$\lambda_{2d}(n)=[\min\{\lambda_1(n-d)+q_1(n-d)-s_1(n-d, n-1), \mu_1(n)-\upsilon_1(n), s_1(n-d, n)-q_1(n-d), \lambda_1(n-d)\}]^+, d=1,\ldots,n, \tag{27a}$$

and when $d=0$, we have

$$\lambda_{20}(n)=[\min\{\mu_1(n)-\upsilon_1(n)-q_1(n), \lambda_1(n)\}]^+. \tag{27b}$$

where $\lambda_{2d}(n)$, $d=0,1,\ldots,n$ are the number of original messages which arrived at Server 1 at time $n-d$ and depart from Server 1 to Server 2 at current time n.

4.2.3. Obtaining retransmission rate $r'_2(n)$

Only the original messages $\lambda_2(n)$ will trigger the retransmissions from Server 1 after a timer times out. Similar to Eq. (2), we can obtain the total generated messages r'_2 for retransmission created by Server 1 for Server 2 at current time slot n as

$$r_2'(n) = \sum_{j=1}^{6} r_{2j}'(n). \tag{28}$$

where r'_{2j} denotes the j^{th} retransmission for the original request messages arriving at Server 2 at time $n-T_j$. We define the total service capacity of Server 2 from time m to n as $s_2(m,n)=\sum_{l=m}^{n}\mu_2(l)$. Like $r_{1j}(n)$, $r'_{2j}(n)$ can be expressed as

$$r'_{2j}(n)=\min\{[\lambda_2(n-T_j)+q_2(n-T_j)-s_2(n-T_j+1,n)]^+, \lambda_2(n-T_j)\}, \tag{29}$$

4.2.4 Obtaining retransmission rate $r_2(n)$

Due to the delay at Server 1, $r'_2(n)$ and $r_2(n)$ are not necessarily equal as mentioned earlier. In order to get $r_2(n)$, we have to characterize the delay at Server 1. According to Queuing-priority Mechanism (c), the newly retransmitted request messages $r'_2(n)$ created by Server 1 for Server 2 enter the queue of Server 1 just after $\lambda_1(n-d)$ and $r_1(n-d)$. Similarly to the approach we used for Section 4.1.2, we can obtain $r_2(n)$ as

$$r_2(n) = \sum_{d=0}^{n} r_{2d}(n), \tag{30}$$

$$r_{2d}(n)=[\min\{r'_2(n-d)+r_1(n-d)+\lambda_1(n-d)+q_1(n-d)-s_1(n-d, n-1), \mu_1(n)-\upsilon_1(n), s_1(n-d, n)-q_1(n-d)-\lambda_1(n-d)-r_1(n-d), r'_2(n-d)\}]^+, d=1,\ldots,n, \tag{31a}$$

$$r_{20}(n)=[\min\{\mu_1(n)-\upsilon_1(n)-q_1(n)-\lambda_1(n)-r_1(n), r'_2(n)\}]^+. \tag{31b}$$

4.2.5. Obtaining response rate $\upsilon_1(n)$

Since each departure request message (i.e., $\lambda_2(n)$ and $r_2(n)$) of Server 2 sends a response message back to Server 1, so we have

$$\upsilon_1(n+1)=\lambda_3(n), \tag{32}$$

where $\lambda_3(n)$ is the departure request messages at Server 2 at current time slot n. Similar to Section 4.1.2, we can obtain $\lambda_3(n)$ as

$$\lambda_3(n)=\sum_{d=0}^{n}\lambda_{3d}(n), \tag{33}$$

$$\lambda_{3d}(n)=[\min\{r_2(n-d)+\lambda_2(n-d)+q_2(n-d)-s_2(n-d, n-1), \mu_2(n), s_2(n-d, n)-q_2(n-d), r_2(n-d)+\lambda_2(n-d)\}]^+, d=1,\ldots,n, \tag{34a}$$

$$\lambda_{30}(n)=[\min\{\mu_2(n)-q_2(n), r_2(n)+\lambda_2(n)\}]^+. \tag{34b}$$

4.3. Generalization of the 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 4.1.2, 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 let 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. A fluid 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 [29].

5. Simulations and performance evaluation

In order to demonstrate the chaotic behaviour of SIP retransmission mechanism, we performed fluid-based Matlab simulation based on the fluid model we have derived. By evaluating the performance of an overloaded tandem server, we not only verified the stability condition provided by Theorem 1, but also investigated whether overload can migrate from server to server.

In Section 4, we did not make any claims about the original message arrival process and server service processes. Therefore they can be any process. This makes our model quite general. However, in order to conduct simulation, we had to choose specific original message arrival processes and server service processes. Measurement results in [25] have shown that throughputs with a SIP server can vary from hundreds to thousands of messages per second. In addition, the processing times for different types of messages may be different [10].

In order to make our simulation results more general and realistic, similar to the experiment in [10], we let the arrival demands follow Poisson distribution and the service time for each type of message be variable with exponential distribution. Within a time slot, the instantaneous service rates vary with the number of original messages, retransmitted messages, and response messages being served at current time slot n, i.e., $\mu_1(n)=\lambda_2(n)+r_2(n)+r_1^d(n)+\upsilon_1(n)$, and $\mu_2(n)=\lambda_2^d(n)+r_2^d(n)$, where $r_1^d(n)$ denotes the departure retransmitted messages at Server 1; $\lambda_2^d(n)$ denotes the departure original messages at Server 2; $r_2^d(n)$ denotes the departure retransmitted messages at Server 2. We can use the similar approach in Section 4.1.2 to obtain $r_1^d(n)$, $\lambda_2^d(n)$ and $r_2^d(n)$. The equivalent service rates measured in the processing rate for original request messages will be $\mu'_1(n)=\lambda_2(n)+\gamma r_2(n)+\beta r_1^d(n)+\alpha\upsilon_1(n)$ and $\mu'_2(n)=\lambda_2^d(n)+\beta r_2^d(n)$, where α, β and γ denote the ratio of the mean processing time of a retransmitted or a response message to that of an original request message. The equivalent service rates $\mu'_1(n)$ and $\mu'_2(n)$ are bounded by the corresponding mean server capacities C_1 and C_2 as measured by their maximal mean service rates for processing original request messages, i.e., $\mu'_1(n) \le C_1$, and $\mu'_2(n) \le C_2$,. The processing time ratios are set as α=0.5, β=1 and γ=1.

In the following results, we consider two typical overload scenarios: (1) Overload at Server 1 due to a demand burst; (2) Overload at Server 2 due to a server slowdown. We have run 10 simulation replications for each scenario to evaluate the performance.

5.1. Overload at Server 1

In this scenario, we investigate the impact of the initial queue size on the SIP overload to verify the stability condition provided by Theorem 1. In order to achieve our goal, a mixture of impulse and step functions is considered for the input demand at Server 1. A demand burst overloaded Server 1 and created an initial queue size at time t=0s. This emulated a short surge of user demands. Normal original request messages entered the Server 1 with Poisson-distributed arrival rate of a mean value λ_1=200 messages/sec. We let the equivalent service rates of Server 1 and Server 2 for the original messages be Poisson distributed too, and the mean service capacities were C_1=1000 messages/sec and C_2=1000 messages/sec respectively. That is, the mean processing times for an original message, a retransmitted message and a response message are 1ms, 1ms and 0.5ms respectively.

The retransmission messages triggered by the overload are redundant messages. Therefore, only the CPU consumed by the original messages and corresponding response messages can be regarded as effective use of resources. We define effective CPU utilization as the ratio of the total mean service rate for the original and corresponding response messages to the service capacity. Considering that a response message corresponds to a original message, the effective CPU utilization for regular user demands is $\rho=(\lambda_1+\alpha\lambda_1)/C_1=(200+0.5\times200)/1000$ =30%.

Since violating the sufficient stability condition of Eq. (8) does not always bring the instability to a SIP server, we would like to investigate how tight the sufficient stability bound for the retransmission mechanism is when a SIP overload happens. We can obtain $j=\lfloor(\mu_1-\upsilon_1-\lambda_1)/\lambda_1\rfloor=3$. Then using Eq. (8), we can obtain the stability condition for the overloaded server as $q_1(0)<\min\{5000, 2783, 2278, 2325\}=2278$ messages. We consider two sub-scenarios with different initial bursty demands which overloaded a single server at time t=0s: (1) a demand burst created an initial queue size as $q_1(0)$=2000 messages < 2278 messages, obeying the stability condition described by Eq. (8); (2) a demand burst created an initial queue size as $q_1(0)$=3000 messages > 2278 messages, exceeding the stability bound (3000−2278)/2278≈30%.

5.1.1. Demand burst of 2000 messages

Figures 5 and 6 show the dynamic behaviour of the overloaded Server 1 when a demand burst created an initial queue size of 2000 messages at time t=0s.

As shown in Fig. 5, the queue size exhibited several sharp surges even though the mean arrival rate of original arrival request messages was smaller than the mean service rate. The reason behind this unusual phenomenon, as shown in Fig. 6, is that the large number of redundant retransmissions were triggered by initial demand burst at retransmission time-out timer, even though the demand resumed the normal rate after an initial demand burst.

Since the initial queue size was below the stability bound, the server could handle the transient overload effectively, thus settling into a stable state eventually. As the buffer became empty after time t≈22s (see Fig. 5), the mean service rate was equal to the mean arrival rate of total messages, much less than mean service capacity (see Fig. 6). When the mean retransmission rate exceeded the mean service capacity (see Fig. 6), the overload server had to use its full capacity to process retransmission messages. Thus no original messages were forwarded to its downstream servers. Consequently, no response messages were sent back to the server temporarily (see Fig. 6).

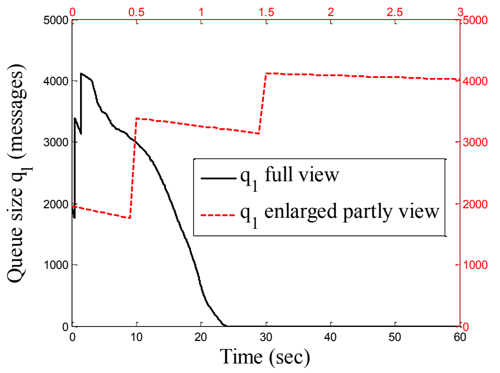


Figure 5. Queue size q_1 (messages) versus time for overloaded Server 1 with initial queue size of 2000 messages.

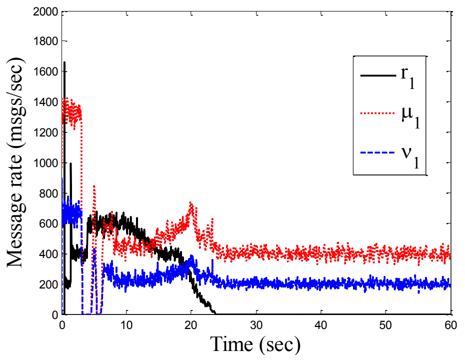


Figure 6. Retransmission rate r_1, service rate μ_1 and response rate υ_1 (messages/sec) versus time for overloaded Server 1 with initial queue size of 2000 messages.

5.1.2. Demand burst of 3000 messages

In this sub-scenario, in addition to verifying that violating stability bound may result in server crash eventually, we have achieved another three goals: (1) showing the accuracy of our fluid model for SIP; (2) demonstrating the benefit of fluid-based simulation over event-driven simulation; (3) demonstrating the impact of varying initial queue size on the standard deviation of simulation replications.

5.1.2.1. Comparisons of fluid and OPNET models

Our fluid model is a slot-by-slot approach which may generate some errors. Event-driven OPNET simulation works on a message-by-message basis which is closer to real system. In order to show the accuracy of our fluid model for the fluid-based Matlab simulation and demonstrate its efficiency, we performed event-driven OPNET simulation in the network topology depicted by Fig. 7. In the OPNET simulator, messages were handled one by one instead of being aggregated over a time slot as in our Matlab simulation. Four user agent clients generated original messages with equal mean rate, and then sent them to a tandem server, as shown in Fig. 7.

The aggregated mean rate of the four user agent clients was set to be equal to the mean original arrival rate in the corresponding Matlab scenario. The mean service rate of the overloaded server is also set to be the same as the mean service rate of the corresponding server in Matlab simulation. The processing speeds of the downstream and upstream servers of the overloaded server are set to be so large that their processing times are negligible. All the sending servers also maintained a list of all outstanding messages for tracking retransmissions. Clearly this list takes a large amount of memory and may cause the simulator to crash when overload happens. Manipulating this list consumes large amount of CPU time and makes OPNET simulation extremely slow. Following the first-come-first-in principle, messages enter the queue in the order they arrive. That is, Queuing-priority Mechanism (c) is unnecessary for OPNET simulation because the chances that two or more messages arrive at exactly the same time are very low and therefore differentiate these messages will make very little difference.

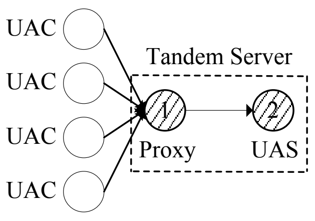


Figure 7. SIP network topology with a tandem server which is marked with diagonal lines.

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 $\bar{X} \pm 1.96 * \sigma / \sqrt{N}$, where $\bar{X}$ and σ are the mean value and the standard deviation of N=10 replications.

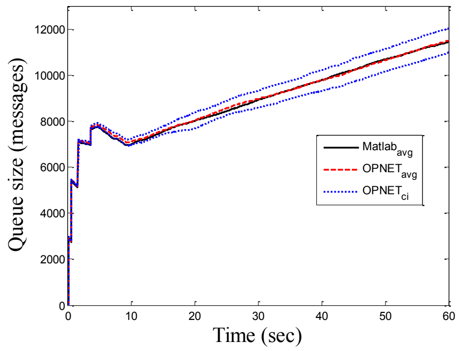


Figure 8. Mean queue size q_1 (messages) of Server 1 and 95% confidence interval versus time for overloaded Server 1 with initial queue size of 3000 messages, where Matlab mean stays insides OPNET 95% confidence interval.

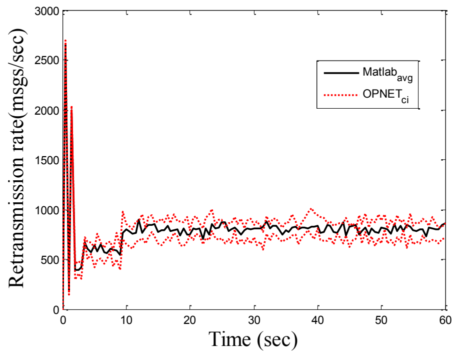


Figure 9. Mean retransmission rate r_1 (messages/sec) for Server 1 and 95% confidence interval versus time for overloaded Server 1 with initial queue size of 3000 messages.

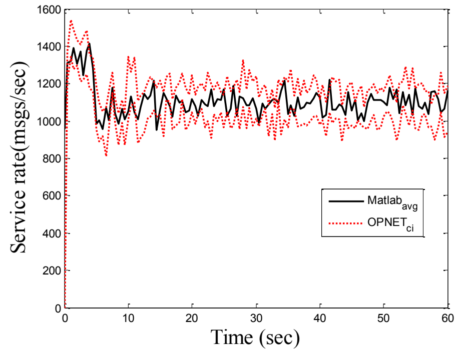


Figure 10. Mean service rate μ_1 (messages/sec) of Server 1 and 95% confidence interval versus time for overloaded Server 1 with initial queue size of 3000 messages.

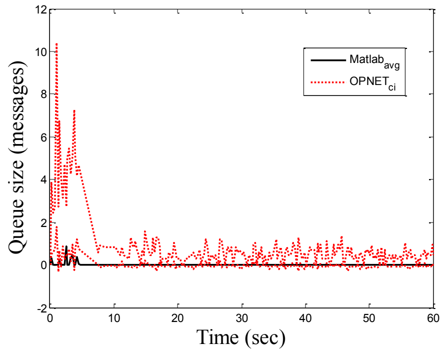


Figure 11. Mean queue size q_2 (messages) of Server 2 versus time in case of overloaded Server 1 with initial queue size of 3000 messages.

Figures 8 to 11 show the dynamic behaviour of the overloaded Server 1 when a demand burst created an initial queue size of 3000 messages at time t=0s. The confidence intervals in Figures 8 to 10 are quite tight compared to their sample means. Also note the sample means of our fluid model and OPNET model overlap nearly perfectly which indicate a good matching between the two types of models, as shown by Fig. 8. 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 Fig.11, the quantization errors take effect as expected due to the small sample means. The maximum quantization error is one or two messages.

The initial demand burst of 3000 messages made Server 1 CPU overloaded immediately (see Fig. 8). Since Server 1 lacked sufficient capacity to process all 3000 messages from demand surge before retransmission timers expire, the redundant retransmissions were triggered and then deteriorated the overload by driving the queue to build up continuously. The extremely long queuing delay corresponding to the extremely large queue size (as shown by Fig. 8) stimulated more subsequent retransmissions (as shown by Fig. 9) with time going. Server 1 had to operate at its full service capacity endlessly (see Fig. 10). Consequently the persistent overload led to the eventual crash of Server 1, well matching the claim by Theorem 1, that is, when the initial queue size exceeded the stability bound, the server had no enough capacity to handle the redundant retransmissions triggered by the transient demand burst.

Given that the mean service capacities of Server 1 and Server 2 were the same, Server 2 maintained almost empty buffer (see Fig. 11).

As the initial queue size created by the demand burst at Server 1 is deterministic for 10 simulation replications, the 95% confidence interval is quite small, as shown in Fig. 8. This indicates that the variation of the Poisson-distributed original message arrival rate and service rate makes a small impact on the queue size of Server 1.

5.1.2.2 Benefit of fluid-based simulation over event-driven 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. Table I shows the simulation time of different mean 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 4 days, while Matlab simulation reduced the simulation time 40,000 times to 8 seconds, as shown by Table I. Therefore, by aggregating events into time slots, fluid-based simulation can reduce the total simulation time significantly.

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

Mean Server Capacity (msgs/sec)	1000	10,000	100,000
OPNET Simulation Time (secs)	128	9,464	341,519
Matlab Simulation Time (secs)	8	8	8

5.1.2.3. Impact of Varying Initial Queue Size

In order to study the impact of varying initial queue size on the standard deviation, a signature effect of chaotic system, we let the initial demand burst uniformly distributed between 1000 messages and 5000 messages, so the mean value of initial queue size becomes 3000 messages. We run 10 replications and 60 replications respectively using Matlab simulation. We use standard deviation to measure the divergence of sample traces with different initial values. Fig. 12 depicts the corresponding mean values and standard deviations of the queue size at Server 1. The randomness of the samples causes small difference in the mean values of 10 replications and 60 replications. However, the standard deviations of both 10 replications and 60 replications are quite large and diverge from their mean values as time evolves. Thus increasing the number of simulation replications cannot prevent the standard deviation from diverging. This kind of divergent phenomenon is a main characteristic of chaotic systems. This validates the chaotic nature of SIP retransmission mechanism.

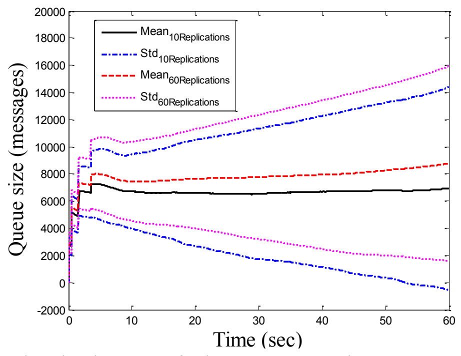


Figure 12. Mean values and standard deviations of the queue size q_1 at Server 1 based on 10 and 60 Matlab simulation replications respectively.

5.1.3. Chaotic nature of SIP retransmission mechanism

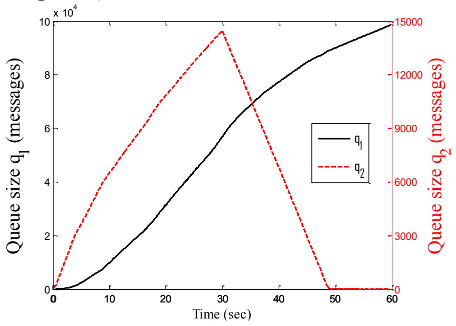
The numerical results in the two sub-scenarios demonstrate the impact of initial queue size on the stability of an overloaded server, where redundant retransmissions are triggered by the overload. A smaller initial queue size less than sufficient stability bound allows the server to cancel the temporary overload, while a larger initial queue size exceeding sufficient stability bound results in infinitely increasing queue size, thus bringing a SIP server to crash even the effective CPU utilization is as low as 30%. This kind of chaotic behaviour is clearly caused by the nonlinear queuing dynamics of the retransmission mechanism.

5.2. Overload at Server 2

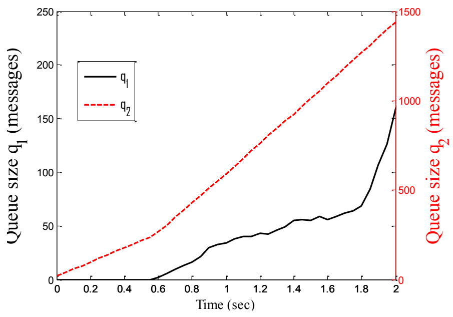
In this scenario, the equivalent service rates of Server 1 and Server 2 for the original messages were Poisson distributed, and the mean server capacities were C_1=1000 messages/sec from time t=0s to t=60s, C_2=100 messages/sec from time t=0s to t=30s, and C_2=1000 messages/sec from time t=30s to t=60s.

The normal SIP traffic entered the Server 1 with Poisson-distributed arrival rate. The mean arrival rates were λ_1=500 messages/sec from time t=0s to t=30s, and λ_1=200 messages/sec from time t=30s to t=60s. We have performed 10 simulation replications and find that 95% confidence interval is quite small, thus we only show the mean value of 10 replications. We will not show the OPNET simulation result because it was very close to the Matlab simulation result as the Subsection 5.1.2.1.

Figures 13 to 15 demonstrate the overload propagation and migration from the downstream overloaded Server 2 to its upstream Server 1. Server 2 became overloaded first, which was followed by a later overload at Server 1 (see Fig. 13), indicating the overload propagation. The queue size at Server 1 increased faster due to the extra work load for handling both retransmissions r_1 and r'_2 (see Fig. 15) for Server 1 and Server 2 respectively. In the mean time, the retransmissions r_1 and r'_2 hindered the new arrival original messages λ_1 from departing Server 1 immediately, thus causing the arrival rate λ_2 of the original request messages at Server 2 to decrease (see Fig. 14).



(a) full view



(b) enlarged partly view

Figure 13. Mean queue sizes q_1 and q_2 versus time upon an initial overload at Server 2.

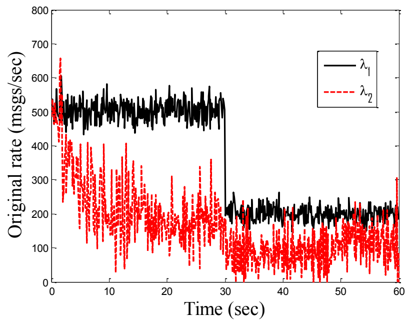


Figure 14. Mean original message rates λ_1 and λ_2 (messages/sec) versus time upon an initial overload at Server 2.

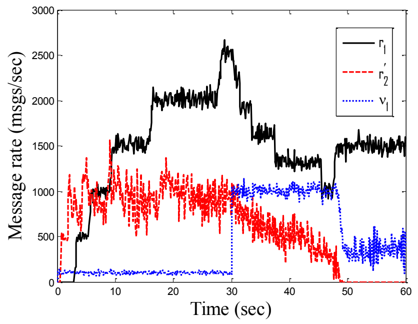


Figure 15. Mean response rate υ_1 and mean retransmission rates r_1 and r'_2 (messages/sec) versus time upon an initial overload at Server 2.

After Server 2 resumed its normal service at time *t*=30s, Server 1 and Server 2 had the same service capacity. As Server 2 processed the residual messages in the queue and fed back response messages to Server 1 more quickly, the response rate of Server 1 increased sharply (see Fig. 15). 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. Therefore the queue at Server 2 started going down. Eventually the overloaded at Server 2 was cancelled while the overload at Server 1 persisted due to its higher arrival rate of aggregated request messages, indicating the overload migration.

5.3. Migration of overload

The two simulation scenarios exhibit different initial overload status in different servers within a tandem server, and both servers have the same CPU processing capacity. We can find that (1) the demand burst at Server 1 produces overload only in Server 1 and will not introduce overload in Server 2; (2) the initial overload at Server 2 is migrated to Server 1 and results in a continuous overload in Server 1, thus bringing Server 1 to crash.

6. Conclusions

We have developed a novel approach to derive a fluid model for a single overloaded server in a SIP network. We use nonlinear difference equations to catch the chaotic behaviour of SIP retransmission mechanism upon SIP overload. Unlike various existing signaling models based on Poisson-distributed arrival rate and service rate, our study considered a general case that both arrival rate and service rate for signaling messages are arbitrarily distributed. Two key breakthroughs of our modeling strategy are (1) derivation of the retransmission messages for the original messages with arbitrary delay and (2) the formulation of the departure process through detailed analysis, which is important for the derivation of the response rate, because the response rate of a upstream server is equal to the departure request message rate of its downstream server, provided that a response message is required for acknowledging each request message [1]. The tandem server has been studied to demonstrate how to apply our analytical approach to create a fluid model for an arbitrary SIP network.

Our fluid modeling and simulation have demonstrated that (1) the similarity between the fluid-based simulation and the event driven simulation for the same network confirms the accuracy of our fluid model; (2) the chaotic characteristic of SIP retransmission makes SIP server very sensitive to an initial queue size caused by a heavy load. Based on the fluid model, a sufficient condition for maintaining system stability has been provided; (3) overload at an downstream server can migrate to its upstream server step by step, thus causing widespread server crashes. Resource overprovision cannot avoid a SIP server crash upon a large queue size introduced by a demand burst or a temporary server slowdown, even its effective CPU utilization is as low as 30%; (4) the root cause of server crashes in a SIP network is redundant retransmissions. This is in sharp difference from traditional congestion problems in IP network which are typically caused by user traffic surges. Existing SIP overload control mechanisms try to copy TCP or load balance schemes which are designed to reduce original message rates and therefore cause revenue loss to the carriers. We propose to reduce the retransmission rate only, while maintaining the original message rate and revenue in case of the overload. A solution can be realized by requiring the downstream server to inform its upstream server about its status, e.g., a new field in the response message to indicate server utilization.

Event-driven simulation requires a series of retransmission timers to track outstanding messages. When overloads happen, messages may build up to drive the number of timers to an extreme value that crashes the simulator eventually. Fluid-based simulation tracks time slot instead of individual timer. Messages arriving within the same time slot will be aggregated and processed together, thus making the computation cost for fluid-based simulation invariant when message arrival rate and server service capacity scale up. When the original message arrival rate and server service capacity are scaled up to 10,000 messages/sec, we have observed two phenomena: (1) our computer only takes 8 seconds to complete the fluid-based Matlab simulation using our fluid model; (2) finishing OPNET simulation requires almost 4 days with the same computer. Therefore, our fluid model can enable the researchers or SIP operator to speed up the SIP performance evaluation using the fluid-based simulation, when extremely high message arrival rate and service capacity of a SIP network are well beyond the computational capabilities of current event-driven simulators.

Acknowledgment

This work was supported by the NSERC grant #CRDPJ 354729-07 and the OCE grant #CA-ST-150764-8.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002.
[2] J. Rosenberg and H. Schulzrinne, *SIP: Locating SIP Servers*, IETF RFC 3263, June 2002.
[3] European Telecommunications Standards Institute, 3rd Generation Partnership Project, Sophia Antipolis, France, 2011, http://www.3gpp.org.
[4] N. Xu and W. Jia, *Joint Packet Scheduling and Radio Resource Assignment for WiMAX Networks*, Ad Hoc & Sensor Wireless Networks, 13(3-4), (2011), pp. 193-208
[5] S.M. Faccin, P. Lalwaney, and B. Patil, *IP Multimedia Services: Analysis of Mobile IP and SIP Interactions in 3G Networks*, IEEE Communications Magazine, 42(1), (2004), pp. 113-120.
[6] J. Rosenberg, *Requirements for Management of Overload in the Session Initiation Protocol*, IETF RFC 5390, December 2008.
[7] E. Noel and C.R. Johnson, *Novel Overload Controls for SIP Networks*, Proceedings of 21st International Teletraffic Congress, Paris, France, September 2009.
[8] R.P. Ejzak, C.K. Florkey, and R.W. Hemmeter, *Network Overload and Congestion: A comparison of ISUP and SIP*, Bell Labs Technical Journal, 9(3), (2004), pp. 173–182.
[9] M. Ohta, *Overload Control in a SIP Signaling Network*, Proceeding of World Academy of Science, Engineering and Technology, Vienna, Austria, March 2006, pp. 205—210.
[10] V. Hilt and I. Widjaja, *Controlling Overload in Networks of SIP Servers*, Proceedings of IEEE ICNP, Orlando, FL, October 2008, pp. 83-93.
[11] C. Shen, H. Schulzrinne, and E. Nahum, *SIP Server Overload Control: Design and Evaluation*, Proceedings of IPTComm, Heidelberg, Germany, July 2008.
[12] A. Abdelal and W. Matragi, *Signal-Based Overload Control for SIP Servers*, Proceedings of IEEE CCNC, Las Vegas, NV, January 2010.
[13] IPTEL Organization, SIP Express Router, open-source project for free SIP/IP Telephony service, 2011, http://www.iptel.org/ser/.
[14] T. Warabino, Y. Kishi and H. Yokota, *Session Control Cooperating Core and Overlay Networks for "Minimum Core" Architecture*", Proceedings of IEEE Globecom, Honolulu, Hawaii, December 2009.
[15] I. Dacosta, V. Balasubramaniyan, M. Ahamad, and P. Traynor, *Improving Authentication Performance of Distributed SIP Proxies*, Proceedings of IPTComm, Atlanta, GA, July 2009.
[16] I. Dacosta and P. Traynor, *Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks*, Proceedings of the USENIX Annual Technical Conference (ATC), Boston, MA, June 2010.
[17] C. Shen and H. Schulzrinne, *On TCP-based SIP Server Overload Control*, Proceedings of IPTComm, Munich, Germany, August 2010.
[18] Y. Hong, C. Huang, and J. Yan, *Mitigating SIP Overload Using a Control-Theoretic Approach*, Proceedings of IEEE Globecom, Miami FL, December 2010.
[19] Y. Hong, C. Huang, and J. Yan, *Controlling Retransmission Rate For Mitigating SIP Overload*, Proceedings of IEEE ICC, Kyoto, Japan, June 2011.
[20] Y. Hong, C. Huang, and J. Yan, *Design Of A PI Rate Controller For Mitigating SIP Overload*, Proceedings of IEEE ICC, Kyoto, Japan, June 2011.

[21] Y. Hong, C. Huang, and J. Yan, *Applying Control Theoretic Approach To Mitigate SIP Overload*, Telecommunication Systems, (2012), in press.

[22] Y. Hong, C. Huang, and J. Yan, *A Comparative Study of SIP Overload Control Algorithms*, in *Internet and Distributed Computing Advancements: Theoretical Frameworks and Practical Applications*, J. Abawajy, M. Pathan, M. Rahman, A.K. Pathan, and M.M. Deris, eds., IGI Global, Hershey, PA, 2012.

[23] Y. Hong, C. Huang, and J. Yan, *Analysis of SIP Retransmission Probability Using a Markov-Modulated Poisson Process Model*, Proceedings of IEEE/IFIP Network Operations and Management Symposium, Osaka, Japan, April 2010, pp. 179–186.

[24] B. Materna, *Threat Mitigation for VoIP*, Proceedings of Third Annual VoIP Security Workshop, Berlin, Germany, June 2006.

[25] E.M. Nahum, J. Tracey, and C.P. Wright, *Evaluating SIP server performance*, Proceedings of ACM SIGMETRICS, San Diego, CA, June 2007, pp. 349–350.

[26] V. Gurbani, V. Hilt, and H. Schulzrinne, *Session Initiation Protocol (SIP) Overload Control*, IETF Internet-Draft, draft-ietfsoc-overload-control-05, October 2011.

[27] K. Alligood, T. Sauer, and J.A. Yorke, *Chaos: an introduction to dynamical systems*, Springer-Verlag, New York, NY, USA, 1997.

[28] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley, *Large-scale network simulation – How big? How fast?*" Proceedings of the IEEE/ACM MASCOTS, October 2003, pp. 116–125.

[29] Y. Liu, F.L. Presti, V. Misra, D. Towsley, and Y. Gu, *Scalable fluid models and simulations for large-scale IP networks*, ACM Transactions on Modeling and Computer Simulation, 14 (3), (2004), pp. 305–324.

[30] D.M. Nicol and G. Yan, *Discrete event fluid modeling of background TCP traffic*, ACM Transactions on Modeling and Computer Simulation, 14 (3), (2004), pp. 211–250.

[31] M. Schwartz, *Broadband Integrated Networks*, Prentice-Hall, NJ, (1996).

[32] D.C. Vasiliadis, G.E. Rizos, C. Vassilakis, and E. Glavas, *Modelling and performance evaluation of a novel internal-priority routing scheme for finite-buffered multistage interconnection networks*, International Journal of Parallel, Emergent and Distributed Systems, 26(5), (2011), pp. 381-397.

[33] Y. Hong, C. Huang, and J. Yan, *Modeling and Simulation of SIP Tandem Server with Finite Buffer*, ACM Transactions on Modeling and Computer Simulation, 21(2), (2011), pp. 11:1–11:27.

[34] Y. Hong, C. Huang, and J. Yan, *Stability Condition for SIP Retransmission Mechanism: Analysis and Performance Evaluation*, Proceedings of IEEE SPECTS, Ottawa, Canada, July 2010, pp. 387-394.

[35] M. Govind, S. Sundaragopalan, K. S. Binu, and S. Saha, *Retransmission in SIP over UDP – Traffic Engineering Issues*, Proceedings of International Conference on Communication and Broadband Networking, Bangalore, India, May 2003.