Evaluation of SIP Proxy Server Performance: Packet-Level Measurements and Queuing Model

Ramesh Krishnamurthy, George N. Rouskas
North Carolina State University, Raleigh, NC 27695-8206 USA

***Abstract*—The growing number of applications that use the Session Initiation Protocol (SIP) to manage media sessions over IP is placing increasing demands on the SIP proxy servers (SPS) that make up the core of the SIP network. In this work we investigate the performance of OpenSIPS, an open source SPS. We have collected a large set of experimental data to characterize the performance of the SPS under various call arrival rates and inter-arrival time distributions. Based on these measurements, we model the SPS as an *M/G/1* queue. A key component of the model is a parameter that captures the cache-miss overhead, i.e., the impact of cache-misses on kernel service times.**

I. Introduction

The Session Initiation Protocol (SIP) is widely used as a signaling protocol for managing media sessions over IP. In this work we investigate the performance of OpenSIPS, an open source SIP proxy server, and make several contributions. (1) We have modified the Linux kernel and the OpenSIPS source code to obtain packet-level measurements for each SIP message, from which the service and waiting times within the kernel and the SIP layer can be easily obtained. (2) We also enhanced SIP*p*, a SIP traffic generator tool, to generate calls with inter-arrival times that follow any user-specified distribution. (3) We have collected a large set of experimental data to characterize the performance of the SPS under various call arrival rates and inter-arrival time distributions. (4) Based on these measurements, we model the SIP proxy server as an *M/G/1* queue. A key component of the model is a parameter that captures the cache-miss overhead, i.e., the impact of cache-misses on socket queue service times.

An analytical model to estimate the mean response time for call setup, as measured from the User-Agent Client (UAC) perspective, was developed in [1]. In this study, the mean response time consists of the processing and queuing delays at the SPS and User-Agent Server (UAS). The SPS and UAS are modeled as a queuing network with six $M/M/1$ queues, each queue corresponding to the processing of one message type at either the SPS or UAS, including failure messages not shown in Figure 2. Based on the observation that the processing of messages of a specific type does not follow an exponential distribution but, rather, it is close to constant, the queuing network of [1] was analyzed in [7] under the assumption that each of the six queues are of the $M/D/1$ type. While these studies provide some insight into the mean response time, the six-node queuing network is not an accurate model of the way the SPS and UAS operate in practice, as typically a *single queue* is used for arriving packets. In [8], the SPS is modeled as a single $M/M/c$ queue, and the analytical results are compared to experimental data collected from an SPS with three threads. This study was further extended in [9]. These studies are based on the fundamental assumption that the service time distribution is exponential. In contrast, we conducted extensive experiments to get an accurate characterization of the service time. Based on our findings, we use a six-modal distribution that provides an accurate representation of service times in the SPS. Further, these earlier studies conducted experiments in a *black-box model* whereby the specifics of packet processing in the kernel and SPS were not considered. On the other hand, we carried out experiments in a *white-box model* by using a measurement methodology designed to capture all components of packet processing within the kernel and SPS.

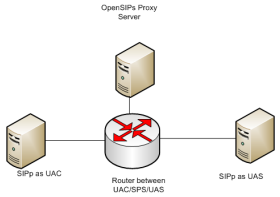

Fig. 1. Testbed for performance measurements of OpenSIPS SPS

We discuss the measurement methodology in Section II, and we present the measurement data in Section III. We develop a queuing model for the SPS proxy server in Section IV, and we conclude the paper in Section V.

II. Measurement Methodology

The main objective of our experimental study is to obtain precise measurements of the time a SIP packet spends within the SPS. Figure 1 shows the network testbed that we used to generate SIP calls and collect measurement data so as to characterize the performance of the SPS as a function of traffic load. The hardware setup in addition to a router, consists of OpenSIPS SPS Server and UAC and UAS. Figure 2 shows the exchange of SIP messages between the UAC and UAS through an SPS, for both the call setup and teardown operations. This is the message flow that we use in our experimental data collection and in modeling the SPS performance.

Let us refer to Figure 3 which illustrates the packet receiving and sending operations within the Linux kernel network stack. Based on this figure, there are three distinct components comprising the processing a SIP packet: packet receiving at the kernel network stack, application layer-SIP packet processing, and packet sending at the kernel network stack.

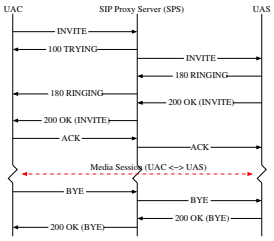

Fig. 2. SIP message exchange for call setup and teardown

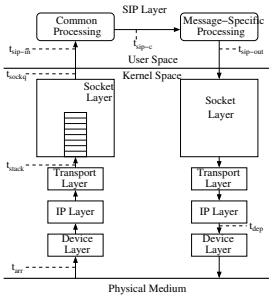

Fig. 3. Path of packets through the Linux network stack and SIP layer

In order to capture the packet service and waiting time components, we modified the OpenSIPS and the Linux kernel source code to log information about each packet as it moves through the system. Each log entry contains a timestamp along with the source IP address, call id, command sequence, and method type of the packet, i.e., all the information necessary to uniquely identify the message type and related call. Timestamps are recorded with microsecond precision at the instants shown in Figure 3. At the end of the experiment, the log file is parsed to determine the various time components.

1) The Kernel Receive K_{rcv} Component: We log three time values for each packet in the kernel and a fourth one as soon as it enters the SIP layer:

- t_{arr}: the time the packet arrives at the SPS, recorded by the kernel on a timestamp attached to the packet at the time it is received by the device layer.
- t_{stack}: the time the packet completes processing by the kernel network stack. We modified the kernel packet structure to add a new field to record this timestamp just before the packet is inserted to the socket queue.
- t_{sockq}: the time at which the application is ready to dequeue the packet from the socket queue. We modified the kernel socket structure to include a new field to record this timestamp.
- t_{sip-in}: the time the packet enters the SIP layer.

We may then calculate K_{rcv} (the time spent within the kernel from the instant the packet is received at the kernel device layer until the instant it is handed off to the SIP layer) and its various components as: $K_{stack} = t_{stack} - t_{arr}, K_{sockq} = K^s_{sockq} + K^w_{sockq} = t_{sockq} - t_{stack}, K_{copy} = t_{sip-in} - t_{sockq}, K_{rcv} = t_{sip-in} - t_{arr}$, where: K_{stack} is the time it takes the packet to undergo processing at the device, network, and transport layers; K^w_{sockq} is the time the packet spends waiting at the socket queue; K^s_{sockq} is the time taken by the kernel to process the packet once it is in the queue, including the time to wake the receiving user level process and handling the dequeue request from the user process; and K_{copy} is the time needed to copy the data from the kernel space to user space.

2) The SIP Service Time T_{sip} Component: T_{sip} reflects the service time of the packet within the SIP layer, and does not include any waiting time. For each packet, we log two additional time values within the SIP layer:

- t_{sip-c}: this is the instant at which the packet processing part that is common to all packet types is complete.
- $t_{sip-out}$: this is the instant at which the SIP layer has completed the processing of the packet and is ready to pass the packet back to the kernel.

From these values, the SIP service time and its subcomponents can be calculated as: $T^1_{sip} = t_{sip-c} - t_{sip-in}, T^2_{sip} = t_{sip-out} - t_{sip-c}, T_{sip} = T^1_{sip} + T^2_{sip}$, where T^1_{sip} is the time spent during common message processing, and T^2_{sip} is the time it takes to perform message-specific processing.

3) The Kernel Sending K_{snd} Component: To determine the time K_{snd} a packet spent traversing the kernel after being processed at the SIP layer, we observed that the call from the SIP layer to send the packet to the kernel returns only when the kernel stack has completed processing and has transferred the packet to the device driver for transmission. We record another timestamp at the SIP layer as soon as this call returns (at which time SIP is ready to fetch the next packet from the socket for processing):

- t_{dep}: the instant at which the packet departs the kernel layer and control is transferred back to the SIP layer; corresponds to the time shown in Figure 3.

We may now calculate the K_{snd} component as: $K_{snd} = t_{dep} - t_{sip-out}$.

III. Experiments and Performance Measurements

We conducted a large set of experiments to measure the components K_{rcv}, K_{stack}, K_{sockq}, K_{snd} and T_{sip} of the time each SIP packet spends at the SPS. For each experiment, the UAC initiates 1M (million) calls to the UAS. Each call is accepted by the UAS, resulting in the message exchange shown in Figure 2. Therefore, for each call, six different messages are generated by either the UAC or the UAS and are forwarded to the other party via the SPS. In other words, for each experiment, the SPS may process up to 6M SIP messages, i.e., up to 1M messages of each type seen in Figure 2. Each experiment is characterized by two parameters:

- *Call arrival rate.* We varied the call arrival rate from 100 cps to 1200 cps. At a rate of 1200 cps, the SPS crashes frequently as it cannot handle the message volume, indicating that the server is severely overloaded.
- *Call inter-arrival time distribution.* The call inter-arrival time is the time between two consecutive `INVITE` messages generated by the UAC. For each call arrival rate, we generated inter-arrival times using exponential and deterministic distributions.

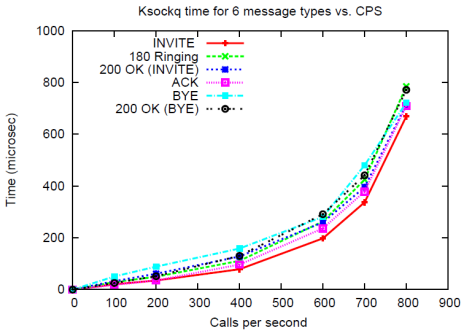

Fig. 4. Mean values of K_{sockq} in the stable region, Poisson arrivals

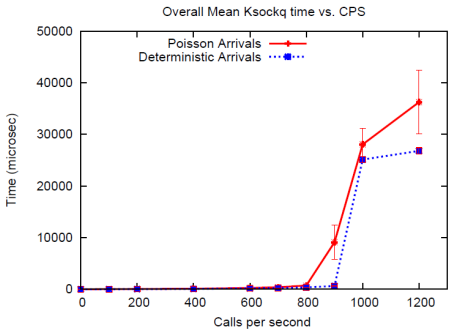

Fig. 5. Overall mean values (in μs) and confidence intervals for K_{sockq}

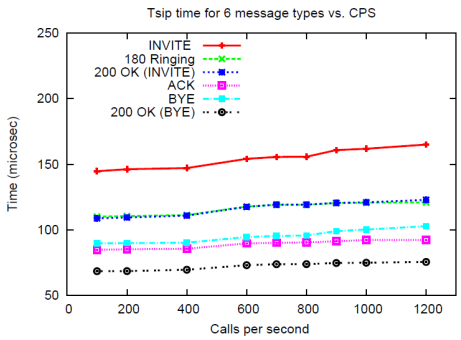

Fig. 6. Mean values for T_{sip}, Poisson arrivals

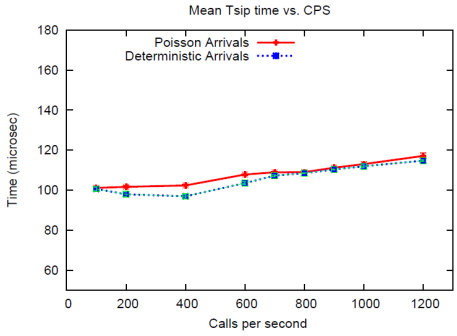

Fig. 7. Overall mean values and confidence intervals for T_{sip}

For each packet processed by the SPS, we log the seven time values t_{arr}, t_{stack}, t_{sockq}, t_{sip-in}, t_{sip-c}, $t_{sip-out}$ and t_{dep}. We then process the log files to obtain the sample mean values for quantities K_{rcv}, K_{stack}, K_{sockq} T_{sip}, and K_{snd}; we record both the overall mean (i.e., across all packets) and the mean per packet type. We use the method of batch means to estimate 95% confidence intervals around the overall mean.

A. Measurement Data for K_{rcv}, K_{stack}, and K_{sockq}

Figure 4 shows the measured values for quantity K_{sockq}, under exponentially distributed inter-arrival times and for various call rates. Figure 5 presents the overall mean value (i.e., averaged over all six message types) for both exponential and deterministic inter-arrivals; all values are expressed in μs. Due to space constraints, we do not present figures of the K_{rcv} and K_{stack} values we have obtained. However, we have observed that kernel stack processing times K_{stack} are largely constant, averaging 2 μs independent of message type and call arrival rate (refer also to Table I for the K_{stack} values obtained through a different experiment). The values of K_{copy}, obtained as $(K_{rcv} - K_{stack} - K_{sockq})$, are also constant at around 2 μs. K_{rcv} values are about 4 μs higher than the K_{sockq} values.

We observe that all mean values of K_{sockq} (and K_{rcv}, not shown here) increase rapidly with the call arrival rate up to 1000 cps, but level off beyond that rate[1]. This increase is due to two factors:

- *Queuing delay.* As the call arrival rate increases, packets arriving at the SPS are buffered at the socket queue and experience increasing waiting times K^w_{sockq} before being delivered to the SIP layer.
- *Cache-Miss overhead.* The number of interrupts increases in direct proportion to the packet arrival rate. Interrupts introduce overhead in the form of the time needed to handle each interrupt, the context-switching operations, and the increase in processing time as a result of cache misses due to these interrupts. A cache miss causes the number of CPU cycles consumed by the network stack receiving process to increase, as the memory access cycles of main memory are several times that of L2 caches. The impact of cache misses on the network stack was measured in [3], [4], [6].

B. Measurement Data for T_{sip}

Figure 6 shows the SIP layer processing times T_{sip} under exponentially distributed call inter-arrival times and for various call rates. Figure 7 presents the mean value over all message types, along with confidence intervals, for both exponential and deterministic inter-arrival times. As we described earlier, SIP processing consists of a common component and a message-specific component, and the differences in the latter component account for the difference in service times among the various message types. For instance, processing an `INVITE` message that initiates a new session requires more operations than other messages (e.g., to verify that this is a

[1]The measured K_{sockq} values level off beyond 1000 cps since, at these high rates, the system saturates and arriving packets are dropped at the kernel. The dropped packets are not observed at the SIP layer where statistics are logged, hence they are not taken into account in the average values shown.

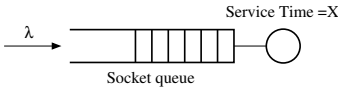

Fig. 8. $M/G/1$ queuing model of the SPS

new transaction and create a new table entry), and this fact is reflected in the data.

Another important observation is that mean SIP service times for all message types increase almost linearly with the call rate. Recall that an SPS operating in stateful mode needs to perform table lookups for each incoming message, so as to match an existing transaction or create a new one. As the call rate increases, the number of transactions in the system also increases, resulting in larger tables at the SIP layer and, hence, longer lookup and overall service times.

C. Measurement Data for K_{snd}

Due to space constraints, we do not present a table with the values of the kernel service time K_{snd} incurred for sending a packet received from the SIP layer down to the device driver. We have observed that this service time varies only slightly for each message type. However, across various call arrival rates, the value remains fairly constant for a given message type. The overall mean value is around 6 μs (refer also to Table I for the K_{snd} values obtained through a different experiment).

IV. $M/G/1$ Queuing Model for the SPS

We now develop an analytical model for predicting the packet-level performance of the SPS, and specifically, the packet waiting time. The experimental data in the previous section indicate that the SPS packet service time distribution exhibits six modal points, corresponding to the six SIP message types. Therefore, we model the SPS as an $M/G/1$ queue [5] as shown in Figure 8. This single queue models the performance of the SPS from the time packets arrive at the socket queue until they depart from the kernel after undergoing SIP processing. From our earlier discussion, the service time X of a packet may be expressed as: $X = K^s_{sockq} + K_{copy} + T_{sip} + K_{snd}$, where K^s_{sockq}, K_{copy} and K_{snd} are the socket, copy and send service times, respectively, in the kernel, and T_{sip} is the service time at the SIP layer.

Despite its simplicity, this model is sufficiently accurate for capturing the performance of the SPS. First, recall that the kernel stack processing times K_{stack} on the receive side are constant around 2 μs across the various arrival rates. Therefore, queuing times at the ring buffer of the device driver are negligible compared to the queuing times at the socket queue and the SIP service time. Hence, we believe that a single queue model accounting for the socket queue is sufficient.

The waiting time for the $M/G/1$ queue is calculated using the well-known Pollaczek-Khinchin formula [5]:

$$W = \frac{\lambda E[X^2]}{2(1-\rho)}. \quad (1)$$

In this expression:

TABLE I
Measured Mean Values (in μs) at 1 cps

Message Type	Pkt Size (B)	K_{stack}	$K^{s,base}_{sockq}$	K_{snd}
INVITE	624	2.6	8.3	6.15
180 Ringing	375	2.2	7.6	5.2
200 OK (INVITE)	542	2.5	8.5	5.5
ACK	466	2.2	7.4	5.6
BYE	466	2.5	8.1	5.9
200 OK (BYE)	367	2.2	7.5	5.2
Overall Mean		2.4	7.9	5.6

- λ is the packet arrival rate, expressed in packets per unit time;
- $\rho = \lambda E[X]$ is the server utilization;
- $E[X]$ is the packet service time at the SPS; and
- $E[X^2]$ is the second moment of the packet service time. Let $E[X_i], i = 1, 2, \ldots, 6$, denote the mean service time of the six SIP message types. Since each call generates exactly six messages, one of each type, the second moment of the service time may be obtained as:

$$E[X^2] = \frac{1}{6}\sum_{i=1}^{6} E[X_i^2]. \quad (2)$$

Therefore, we use the estimates of $E[X_i]$ and $E[X_i^2]$ from the measurement data to obtain the mean packet waiting time from expression (1).

A. Estimating the K^s_{sockq} Component of the Service Time X

The service time of a packet consists of the four components discussed earlier. In our experiments, we have measured directly three of the components, K_{copy}, T_{sip}, and K_{snd}. The fourth component, K^s_{sockq}, represents the processing that is incurred by the packet from the instant, t_{stack}, it is added to the socket queue until the instant, t_{sockq}, it is removed from the queue (see Figure 3). K^s_{sockq} is one of the two components of K_{sockq}; the other component, K^w_{sockq}, represents the waiting time of the packet at the socket queue. Although we have directly measured K_{sockq}, it is important to have an accurate estimate of K^s_{sockq} (and, consequently, K^w_{sockq}) to apply the $M/G/1$ model. To this end, we first obtain a baseline measurement of K_{sockq} under conditions of no queuing, and then we adjust this baseline value for higher call rates by accounting for the overhead caused by cache misses.

1) The K_{sockq} Component Under No Queuing: We conducted a separate experiment to obtain the time values t_{arr}, t_{stack}, t_{sockq} and t_{sip-in} under conditions that ensured no queuing at the kernel socket as the packets move through the network stack on the way to the SIP layer. Let us denote this quantity as $K_{sockq-noq}$. We use this quantity as the baseline value $K^{s,base}_{sockq}$, i.e., $K^{s,base}_{sockq} = K_{sockq-noq}$.

To measure the $K_{sockq-noq}$, we generated SIP calls at a rate of 1 cps, since at this rate packets belonging to different calls do not interfere with each other; The results are shown in Table I. We observe that there is little difference in the kernel processing time across the six message types.

TABLE II
WAITING TIMES (IN μS): MEASURED VS. ANALYTICAL, EXPONENTIAL DISTRIBUTION INTER-ARRIVAL TIMES

Model Parameters	Call Arrival Rate								
	100cps	200cps	400cps	600cps	700cps	800cps	900cps	1000cps	1200cps
λ (packets/sec)	600	1200	2400	3600	4200	4800	5400	6000	7200
$\alpha(\lambda)$	3.0	5.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0
$E[X]$	133.4	149.88	174.53	179.98	181.09	181.185	183.49	185.23	189.5
$\rho = \lambda E[X]$	0.080	0.1798	0.4188	0.6479	0.7606	0.8698	0.9908	1.111	1.3644
W (model)	6.0	16.89	64.21	169.19	293.98	617.77	10171.9	N/A	N/A
W (measured) $= K^w_{sockq}$	5.485	13.59	53.27	190.92	333.52	664.57	9049.07	27997	36194

However, it is not reasonable to set the value of K^s_{sockq} in the expression for the packet service time equal to the baseline value $K^{s,base}_{sockq}$ shown in Table I, as doing so would result in an underestimation of the actual K^s_{sockq} values under higher call arrival rates. Specifically, at 1 cps there is no overhead due to cache-miss, whereas as the call arrival rate increases, this overhead becomes substantial, as we discussed earlier. The cache-miss overhead effectively increases the service time of each packet within the socket queue, and it must be taken into account explicitly in order to arrive at an accurate estimate of the overall packet service time to be used in the waiting time formula (1). The cache-miss primarily impacts the socket-queue service time, as the packet is waiting in queue, the data and instruction related to the queue and packet, is not in active use by the processor, and the cache-replacement algorithm is most likely flushing this data from cache. For all other service time, the processor is actively executing the instructions, as a result, there would be minimal cache replacement.

2) Modeling the Cache-Miss Overhead: From basic computer architecture principles [2], the execution time of an operation is given as the product of (number of instructions) $\times$ (cycles per instruction) $\times$ (time per cycle). Interrupts pollute the cache, increasing cache misses, and in turn increasing the number of cycles per instruction; the other two values in the product are constants for a given operation. We model this cache-miss overhead by expressing the K^s_{sockq} service times as a function of the baseline value $K^{s,base}_{sockq}$ and the server utilization ρ, as follows:

$$K^s_{sockq}(\lambda) \;=\; \alpha(\lambda)\; K^{s,base}_{sockq}. \qquad (3)$$

In the above expression, parameter α, given as a function of λ, adjusts the service time $K^{s,base}_{sockq}$ under no queuing delay (i.e., under minimal cache-miss overhead) to account for the cache misses due to interrupts under a given packet arrival rate λ. $E[X]$ values is then updated to be $E[X] = K^s_{sockq}(\lambda) + K_{snd} + K_{copy} + T_{sip}$.

Based on our experimental results, we model parameter $\alpha(\lambda)$ as a piece-wise linear function of the arrival rate λ, expressed in units of packets/sec, such that: $\alpha(0) = 1$, $\alpha(600) = 3$, $\alpha(1200) = 5$, and $\alpha(\lambda) = 8$ for $\lambda \geq 2400$. This function reflects our observations that (1) as the packet arrival rate increases, the cache becomes more polluted resulting in higher memory access times, (2) the marginal rate at which cache pollution increases diminishes with increasing packet arrival rates, and (3) after a point, the cache will always be polluted, hence there would be no further deterioration due to further increases in the packet arrival rate.

Using the values for α from this function, Table II compares the measured waiting times to the ones obtained through the $M/G/1$ model. We observe that there is a good match between analytical and measured values in the stable region, i.e., up to 800 cps. At arrival rates of 900 cps or higher, the system becomes unstable, losses increase sharply, hence the $M/G/1$ model is not valid. In fact, in this overload region, the estimated value of ρ is higher than 1, hence the Pollaczek-Khinchin formula (1) cannot be applied.

Despite their simplicity, the $M/G/1$ and cache-miss overhead models are sufficiently accurate in two aspects that are important to service providers: jointly they (1) correctly predict the measured packet waiting times at the socket queue within the stable region, and (2) accurately predict the transition to the overload region through the value of ρ. Using this model, service providers only need to monitor the packet arrival rate λ at the SPS to be able to estimate the packet waiting times, as well as detect whether congestion is imminent.

V. CONCLUSIONS AND FUTURE WORK

We have carried out a large set of experiments to characterize the performance of the OpenSIPS SPS as a function of call arrival rate. We have also presented an $M/G/1$ model of the SPS that takes into account the cache-miss overhead. In our ongoing research, we are investigating the impact of multiple cores on the performance of the SPS and other applications.

REFERENCES

[1] V.K. Gurbani, L. Jagadeesan, and V.B. Mendirittam. Characterizing the Session Initiation Protocol (SIP) network performance and reliability. *ISAS 2005*, pages 196–211, April 2005.

[2] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Analysis*. Morgan Kaufmann, 3rd edition, 2003.

[3] H-Y. Kim, S. Rixner. Performance characterization of the FreeBSD network stack. http://www.cs.rice.edu/CS/Architecture/docs/kim-tr05.pdf.

[4] F. Liu *et al. Characterizing and Modeling the Behavior of Context Switch Misses.* ACM PACT, October 2008.

[5] L. Kleinrock. *Queueing Systems, Volume 1*. John Wiley & Sons, 1975.

[6] E. Nahum *et al.*. Cache behavior of network protocols. *ACM SIGMETRICS*, vol. 25, pp. 169–180, June 1997.

[7] S.V. Subramanian and R. Dutta. Comparative study of M/M/1 and M/D/1 models of a SIP proxy server. *ATNAC 2008*

[8] S.V. Subramanian and R. Dutta. Measurements and analysis of M/M/1 and M/M/c queueing models of the SIP proxy server. *ICCCN 2009*,

[9] S.V. Subramanian and R. Dutta. Performance and scalability of M/M/c based queueing model of the SIP proxy server - a practical approach. *ATNAC 2009*,