Performance Evaluation of Multi-Core, Multi-Threaded SIP Proxy Servers (SPS)

Ramesh Krishnamurthy[†], George N. Rouskas[†⋆]
[†]North Carolina State University, Raleigh, NC 27695-8206 USA
[⋆]King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—Process schedulers are part of the core functionality of an operating system (OS), and have been enhanced over the years to account for multiple cores in the processors and to support multi-threaded applications. In this study, we investigate the impact of the Linux scheduler's load-balancing algorithm on the performance of multi-threaded OpenSIPS (an open source SIP proxy server, SPS) running on a multi-core processor system. Linux uses the "completely fair scheduler" (CFS) scheduling policy and provides parameters specifically tunable in a multi-core environment. We conducted extensive experiments and analyzed the collected data to characterize the performance of SPS as a function of the number of CPU cores, the number of server threads and the call arrival rate. Based on our analysis, we show how to configure the various scheduler parameters as a function of the number of CPU cores to achieve a significant improvement in SPS performance. We further present a capacity planning model as a tool that service providers may use to obtain a first-order approximation of the capacity of their system that yields a good match to experimental results.**

I. Introduction

Most multi-core processors in use currently follow the symmetric multi-processor (SMP) paradigm, whereby all cores are identical, are controlled by a single instance of the OS, and share a common main memory. With SMP, processes that do not need to share data with each other may be run on independent CPU cores to improve the performance of each process. The OS scheduler performs load balancing so as to ensure that some cores do not become overloaded if other cores have available processing capacity. Given the proliferation of multi-core systems, the characterization of the performance of multi-threaded applications on such systems is of practical importance.

The scalability of Linux on a multi-core system was analyzed in [1] by examining seven system applications. It was determined that all applications except one trigger a scalability bottleneck in the Linux kernel, and several modifications to the kernel were introduced to reduce this bottleneck. In [2], the scalability of a multi-core web server was examined, and it was observed that the capacity of the address bus in the eight-core system was the limiting factor in performance scaling. The performance of a SIP server on multi-core systems was studied in [3]. This study analyzed the performance of a realistic SIP workload on three different multi-core architectures and suggested improvements to certain operations, including garbage collection and lock contention, to improve performance.

Several studies have specifically investigated the performance of SIP proxy servers (SPS). A load balancing algorithm for processing SIP messages in server clusters was developed in [4] and led to improvements in response time. The deployment of a multimedia service involving SIP sessions and MGCP connections was studied in [5], and strategies consisting of resource allocation and configuration in a virtualized environment were proposed to provide an optimal deployment. There have also been several attempts in the literature to characterize the performance of the SPS analytically, including [6]–[8]; these studies make specific assumptions about the service time and its distribution. On the other hand, in our earlier work [9], [10], we have carried out comprehensive packet-level measurements to obtain an accurate characterization of SPS performance in terms of service time, waiting time and packet drop rate.

In this paper, we investigate the impact of the load-balancing parameters of the Linux completely fair scheduler (CFS) policy on the performance of multi-threaded SPS on multi-core systems. Our work differs from the above studies in that we specifically focus on the Linux CFS scheduler, and we provide guidelines for configuring the scheduler to optimize load-balancing among the CPU cores. Following the introduction, we describe the experimental testbed measurement methodology in Section II. In Section III, we identify and tune the CFS scheduler parameters for multiple cores, and present experimental performance data in Section IV. In Section V, we develop a capacity planning model for a multi-core multi-threaded SPS system. We conclude the paper in Section VI.

II. Measurement Methodology and Experiments

SPS is a central component of the SIP infrastructure, and handles all SIP messages generated by the user-agent client (UAC) and user-agent server (UAS) during setup, modification or termination of each media session. In case of overload, the SPS may become a performance bottleneck that limits the ability of users to establish SIP sessions. Consequently, we only focus on the performance of SPS in this work.

We use the same testbed consisting of an OpenSIPS SPS and SIPp UAC and UAS, as in our earlier work [9], [10]. For the performance evaluation of multi-threaded SPS on multiple cores, we upgraded the OpenSIPS SPS server to use Linux OS version 3.2.51 that is run on the system with two quad-core Intel Xeon CPU E5540 @ 2.53 GHz processors of Figure 1.

We modified the OpenSIPS and Linux kernel source code, and designed a set of experiments to obtain exact time measurements for each SIP packet in the SPS, from the arrival

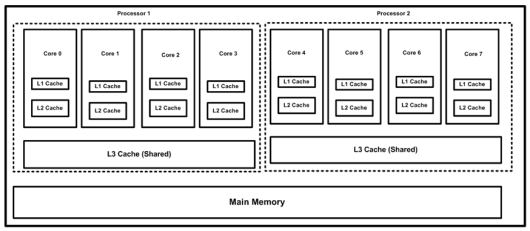


Fig. 1. Dual quad-core processor hosting the OpenSIPS server

instant at the kernel to the departure instant from the kernel after the packet has undergone processing at the SIP layer, as described in [9], [10]. Specifically, we measure two main time components for each packet, the kernel receive time, K_{rcv}, and the SIP processing time, T_{sip}. K_{rcv} represents the time a packet spends within the kernel after arrival, and consists of the time the packet takes to undergo processing at the device, network, transport and socket layers, the waiting time at the socket queue, and the time for the packet to be copied to user space from kernel space. T_{sip} denotes the time the packet undergoes processing within the SIP layer. As we explain in [9], K_{rcv} and T_{sip} represent the waiting and service times of a packet, respectively.

For each experiment, 100,000 calls were started between the UAC and UAS via the SPS. For each call, the messages exchanged between the UAC and UAS are the SIP call setup messages INVITE, 180 RINGING, 200OK and ACK, and the call teardown messages, BYE and 200OK. For each message processed by the SPS, we measured the time components described above to determine the waiting and service times of the message through the SPS; we also kept a count of any messages dropped. Each experiment was characterized by three parameters: *(1) Number of CPU cores.* The experiments were conducted with the SPS server configured as a 2-, 4-, or 6-core system. *(2) Number of server threads.* Experiments were conducted with 2, 4, 6, 8, and 16 server threads for each of the multi-core systems above. *(3) Call arrival rate.* We varied the call rate starting at 200 calls per second (cps), with an increment of 200 cps, up to a maximum call rate beyond which the SPS is overloaded and the drop rate exceeds a certain threshold.

Upon completion of an experiment for a specific call rate and number of cores and server threads, we process the logged data and calculate the sample mean values for K_{rcv} and T_{sip}; we obtain these mean values for each SIP message type as well as the overall mean across all six message types. We also estimate 95% confidence intervals around the overall mean. In addition, we use the `netstat` command to measure the packet drop rate.

III. Impact of Process Scheduler on Performance

In the 3.0-based Linux version, two configurable scheduler parameters are available specifically for tuning the multi-core operation and performance of the system. `sched_migration_cost` is a tunable parameter used to specify the "cost" of migrating a task from the current CPU to a CPU that is becoming idle. The scheduler load-balancing algorithm uses this parameter to allow a CPU going idle to pull tasks from another CPU. The `sched_tunable_scaling` parameter allows the various scheduler parameters to be *scaled* as a function of the number of CPUs in the system. There are three options for this parameter: "no scaling" (i.e., the values of other scheduler parameters are used without modification); "logarithmic scaling" (i.e., other parameter values are multiplied by $1+\log(ncpus)$); and "linear scaling" (i.e., parameter values are multiplied by $ncpus$).

The default value for `sched_migration_cost` is 500,000 (i.e., 500 μs), and for `sched_tunable_scaling` is "logarithmic scaling." However, the study in [11] found that setting the value of `sched_migration_cost` to 5,000,000 (5 ms) instead of the default one results to better performance. This is due to the fact that specifying a very high migration cost forces the scheduler to keep a task in the current CPU, thereby increasing cache utilization.

In our earlier work [10] on a single-core system, we used the following values for three other scheduler parameters to move the scheduler policy to "server" mode, as recommended in [12]: `sched_latency_ns` = 1,000,000 (1 ms), `sched_min_granularity_n` = 100,000 (100 μs), and `sched_wakeup_granularity_ns` = 25,000 (25 μs). In this study, we denote as the *baseline* "multi-core server" mode the scheduler configuration in which, in addition to the above three parameter values, the values of the multi-core specific parameters are set to: `sched_migration_cost` = 5,000,000 (5 ms), and `sched_tunable_scaling` is "logarithmic scaling."

A. Enhanced Multi-Core Server Mode

We note that the parameter value recommendations in [12] are for a generic server configuration and do not take into account workloads or operation characteristics specific to SPS. In this section, we develop a methodology for configuring the scheduler parameters specifically for SPS servers.

According to [13], the two primary factors operators consider when designing their network is the service availability to end-users and the cost of operating the network. Furthermore, from an economic standpoint, network operators aim to achieve high server utilization in order to maximize the return on their capital investment. These observations motivate us to develop guidelines for tuning the scheduler parameters so as to balance these two conflicting objectives: availability of SIP service and SPS server utilization. To illustrate the tradeoffs involved, consider the scheduler parameter `sched_migration_cost`. Increasing the value of this parameter considerably higher than the default value of 500,000 (500 μs), e.g., as suggested in [11], will allow SPS threads to remain resident in a single CPU. However, doing so may cause the CPU to become overloaded as the call arrival rate increases, which in turn will result in excessive loss of SIP call setup packets and negatively affect service availability. Therefore, we consider the packet drop rate (PDR) as the key

performance metric of interest. Note that the only packets seen by the SPS are SIP call setup and teardown messages. Since the loss of any of these messages affects the call establishment process, we argue that the PDR captures the impact of server overload on end-user experience.

We use the `netstat` command in each experiment to obtain the number of SIP messages dropped at the SPS; this number is provided by the `RcvErrors` counter that is incremented by the Linux kernel when the receive buffer is full (note that in our experiments only the SIP process is active, hence the `RcvErrors` counter provides an accurate count of dropped SIP messages). Therefore, we estimate the PDR metric as: `PDR` = `RcvErrors/Total SIP MSGs`.

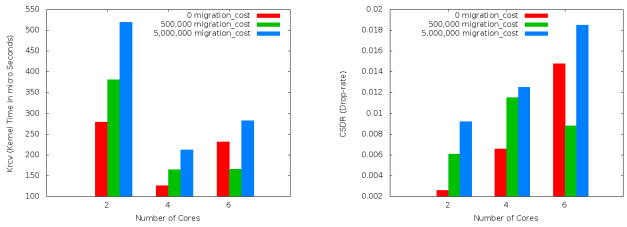
We now summarize our findings regarding the impact of each scheduler parameter. (1) $sched\_latency\_ns$: Setting this parameter to the fixed value 800,000, independent of the number of threads, achieved the best results for the single-core SPS system [10]. In experiments with multi-core systems, scaling this value linearly with the number of CPU-cores resulted in the best performance in terms of PDR. Therefore, we used the values $800,000 \times ncpus$, where $ncpus = 2, 4, 6$, for the 2-, 4-, and 6-core systems, respectively. (2) $sched\_min\_granularity\_ns$: As in [10], we set this parameter to a quantity that corresponds to the measured value of mean service time T_{sip} at the point where the system starts experiencing overload. Note that the minimum value allowed for this parameter is 100,000. The specific values we used for this parameter were as follows: *2-core system:* 100,000 for 2, 4, and 6 threads, 150,000 for 8 threads, and 200,000 for 16 server threads; *4- and 6-core systems:* 100,000 for 2, 4, 6 and 8 threads and 200,000 for 16 server threads. (3) $sched\_wakeup\_granularity\_ns$: Setting the value of this parameter to zero achieved the best results across all threads and CPU core configurations [10]. (4) $sched\_migration\_cost$: For 2- and 4-core systems we set this parameter to zero, while for 6-core systems a value of 500,000 provided the best results. In the next section we present experimental results that justify this choice of values. (5) $sched\_tunable\_scaling$: Recall that this parameter may be used to scale the values of other scheduler parameters either logarithmically or linearly with the number of cores. Since our findings indicate that there is no common scaling factor for the other four parameters above, we set this parameter to "no scaling."

We will refer to the configuration of CFS with these parameter values as the *enhanced* "multi-core server" mode.

IV. Experimental Results

A. Impact of `sched_migration_cost`

Figure 2 shows the impact of varying the value of $sched\_migration\_cost$ on the packet waiting time (K_{rcv}) and PDR, as a function of the number of CPU cores. In these experiments we used the following values for the call arrival rate and number of threads: 2-core system – 4200 cps, 4 threads; 4-core system – 5400 cps, 6 threads; 6-core system – 6200 cps, 8 threads. Results for other call arrival rate and thread values



(a) (b)

Fig. 2. Impact of $sched\_migration\_cost$ on (a) K_{rcv} (b) PDR

are very similar and are omitted. Also, the scheduler was configured in the enhanced "multi-core" mode described in the previous section, in that all parameters were set to values dictated by that mode, except the $sched\_migration\_cost$ parameter whose value was varied as shown in the figures.

Consider the CPU architecture shown in Figure 1, and recall that the value of $sched\_migration\_cost$ is used by the load balancing algorithm to determine whether to move tasks to an idle CPU, with a low value allowing an idle CPU to pull tasks more easily. Also note that the main operation of the SPS involves processing a packet and forwarding it to the UAC or UAS. For 2- and 4-core systems, a value of zero provided the best results: in these systems, all cores are part of same processor, task migration incurs minimal cache penalty, hence setting this cost to zero allows for better load balancing. For the 6-core system, the most effective value is 500,000. In such systems, the cores are distributed across two processors, thus the penalty of migrating the task in terms of cache miss is, on average, higher. Therefore, using a low but non-zero value for the migration cost in an attempt to keep tasks on the same CPU provided an appropriate balance between the cost of cache misses and load balancing across CPUs. In all cases, setting the migration cost to the high value recommended in [11] results in low or no thread migration; hence, increased load in a single core leads to high packet drop rates even if other CPUs have available capacity. These results indicate that scheduler configuration is highly application-dependent and settings that work well for some applications may result in poor performance for others.

B. SPS Performance

Tables I and II present the measured values of T_{sip} (service time of SIP messages), K_{rcv} (waiting time), and PDR under the baseline and enhanced multi-core server mode, respectively, for the 2-core system and various server thread configurations. Tables IV and V present the same data for the 4-core system, while Tables VII and VIII show data for the 6-core system. Each column in these values presents the average of thirty experiments for the stated number of threads and load (in calls per second, cps). For each value of the number of threads, we present results for two load values: the value at which the PDR exceeds 1% for the first time, and the immediately lower value.

We make two observations. For the same load value, configuring the scheduler parameters to the enhanced server mode

always improves the SPS performance over the baseline mode, in terms of PDR and waiting time. Furthermore, in all cases the load at which the PDR rate crosses the 1% threshold we imposed is higher for the enhanced mode than the baseline mode in terms of CPS. We also see that, as the number of cores increases, this threshold load is higher for 4-core compared to 2-cores, and increases further for 6-core; this result is expected given the higher capacity available with additional cores.

To better illustrate the performance improvement under the enhanced mode, Tables III, VI, and IX compare the PDR and K_{rcv} values for 2-, 4-, and 6-core systems, respectively. The results shown are for a specific load for each thread value; this load value corresponds to the scenario where the PDR rate under the baseline mode exceeds the 1% threshold for the given number of threads. As we can see, the performance improvement is between 4-53% for the PDR rate, and between 1-72% for K_{rcv} for the 2-core system. For the 4-core (respectively, 6-core) system, the improvement is 14-42% (respectively, 33-58%) for the PDR and 2-42% (respectively, 12-45%) for K_{rcv}. In fact, the multi-core enhanced server mode shows improvements across all core and thread configurations.

V. Capacity Planning Model

Several studies have investigated aspects of capacity planning. A new methodology was presented in [14] for the efficient analytic solution to account for the burstiness of workload so as to develop a model for capacity planning. In [15], time-series analysis techniques were used to automatically adjust the number of users for an on-demand streaming service and the server bandwidth demand; the proposed mechanism was evaluated on a dataset collected from a video-on-demand service provider. The study in [16] compared the static vs. dynamic resource allocation of virtual machines (VMs) in corporate clouds to evaluate the energy efficiency of each mechanism. The authors concluded that dynamic resource allocation and associated migration overhead may cost more than static VM allocation and does not increase energy efficiency.

Based on the PDR measurements we have collected from our experiments, we now develop a capacity planning model that cloud providers and service providers may use to obtain a first-order approximation of the load (in calls per second) that can be supported by their SPS servers without exceeding the 1% drop rate threshold. Referring to Figure 1, let p and c be the number of independent processors and CPU cores, respectively, in the system that are available to run SPS threads. Also, let $T_{p,c}$ be the number of threads that provides the best performance (i.e., the highest call arrival rate for which the PDR under the enhanced multi-core server mode does not exceed 1%) for the given number of CPU cores and processors. In our experiments, we have found that in systems with only $p = 1$ processor with c cores, setting $T_{1,c} \approx c + 2$ provides a good balance between the conflicting factors of cache locality, multi-threading overhead and process migration cost, and provides the best results. However, for $p \geq 2$ of processors whereby threads execute on cores that are in different processors, keeping the number of threads close to the total number of cores provides the best performance as it avoids the additional migration cost of moving threads from one processor to another. In this case, we let $T_{p,c} \approx c, p \geq 2$. Indeed, referring to Tables II, V, and VIII, we see that the above formula accurately predicts the number of threads that gives the best results for the 2-, 4-, and 6-core systems as 4, 6, and 6 threads, respectively (note that $p = 1$ for 2- and 4-core systems, while $p = 2$ for the 6-core system).

Now let $C_{1,1}$ be the baseline capacity of a single-core, single-thread system; in our earlier studies [9], [10], we established $C_{1,1} = 1000$ cps for our system. Then, the capacity of the multi-core, multi-threaded system can be estimated as: $C_{p,c} = T_{p,c} \times C_{1,1}$. Applying this expression provides estimates of the CPS capacity of 4000 cps for the 2-core system and 6000 for the 4- and 6-core systems, a good first-order approximation of the experimental results.

VI. Concluding Remarks

We investigated the performance of multi-core, multi-threaded SPS and the impact of Linux CFS scheduler tuning on SPS in terms of packet drop rates and waiting times. We have determined CFS scheduler settings that result in significant gains in performance, so as to extract performance gains from the existing computing infrastructures without additional capital expenses. We further developed a capacity planning model that provides a good first-order approximation of the total capacity of the SPS system in terms of the call arrival rate that may be supported without affecting user experience.

References

[1] S. Boyd-Wickizer, *et al*. An analysis of linux scalability to many cores. *USENIX OSDI*, 2010.

[2] B. Veal and A. Foong. Performance scalability of a multi-core web server. *3rd ACM/IEEE ANCS*, pp. 57–66, 2007.

[3] C.P. Wright, *et al*. SIP server performance on multicore systems. *IBM Journal of Research and Development*, 54(1):7:1–7:12, January 2010.

[4] H. Jiang, *et al*. Design, implementation, and performance of a load balancer for SIP server clusters. *IEEE/ACM Trans. Netw.*, 20(4):1190-1202, Aug. 2012.

[5] M. Femminella, *et al*. Optimal deployment of open source application servers providing multimedia services. *IEEE Network*, 28(5):54-63, 2014.

[6] S.V. Subramanian and R. Dutta. Comparative study of M/M/1 and M/D/1 models of a SIP proxy server. *ATNAC 2008*, pp. 397-402, Dec. 2008.

[7] 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*.

[8] 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*, pp. 1-6, Dec. 2009.

[9] R. Krishnamurthy and G. N. Rouskas. Performance evaluation of multi-threaded SIP servers. In *Proceedings of IEEE ICC 2015*, June 2015.

[10] R. Krishnamurthy and G. N. Rouskas. Evaluation of SIP proxy server performance: Packet-level measurements and queuing model. In *Proceedings of IEEE ICC 2013*, pages 2330–2336, June 2013.

[11] CFS Load Balancing for SQL. http://tinyurl.com/CFS-SQL-Load.

[12] CFS Tuning by IBM. http://tinyurl.com/CFSTuning.

[13] Capacity Management and Optimization of Voice Traffic. https://www.cisco.com/en/US/technologies/tk869/tk769/technologies_white_paper0900aecd8070329d.html.

[14] G. Casale, N. Mi, and E. Smirni. Model-driven system capacity planning under workload burstiness. *IEEE Trans. Comp.*, 59(1):66-80, Jan. 2010.

[15] D. Niu, *et al*. Demand forecast and performance prediction in peer-assisted on-demand streaming systems. *INFOCOM*, pp. 421-425, 2011.

[16] A. Wolke, *et al*. Planning vs. dynamic control: Resource allocation in corporate clouds. *IEEE Transactions on Cloud Computing*, 2015.

TABLE I
MEASURED SPS PERFORMANCE, BASELINE MULTI-CORE SERVER MODE, SPS ON 2-CORE

Model Parameters	Number of Server Threads									
	2 Threads		**4 Threads**		**6 Threads**		**8 Threads**		**16 Threads**	
	3000cps	3200cps	3600cps	3800cps	3400cps	3600cps	3000cps	3200cps	2400cps	2600cps
Arrival rate (packets/sec)	18000	19200	21600	22800	20400	21600	18000	19200	14400	15600
T_{sip} (μs)	58.05	60.31	74.92	78.22	96.78	104.48	114.64	124.6	204.04	218.29
K_{rcv} (μs)	474.97	684.87	292.39	373.72	319.04	436.66	308.36	412.81	352.09	395.01
Call-setup Drops	2762	4202	2206	3792	2883	3614	2850	4452	2804	3364
Call-setup Messages	327213	322174	322212	318275	323412	320313	328611	323481	339835	335899
Total Messages	390029	379763	373528	366842	379274	372628	394732	384077	427449	417470
PDR	0.0084	**0.0130**	0.0053	**0.0119**	0.0089	**0.0113**	0.0087	**0.0137**	0.0083	**0.0100**

TABLE II
MEASURED SPS PERFORMANCE, ENHANCED MULTI-CORE SERVER MODE, SPS ON 2-CORE

Model Parameters	Number of Server Threads									
	2 Threads		**4 Threads**		**6 Threads**		**8 Threads**		**16 Threads**	
	3800cps	4000cps	4400cps	4600cps	3800cps	4000cps	3200cps	3400cps	2600cps	2800cps
Arrival rate (packets/sec)	22800	24000	26400	27600	22800	24000	19200	20400	15600	16800
T_{sip} (μs)	54.41	53.55	83.62	85.34	101.28	116.15	125.78	139.26	218.07	262.32
K_{rcv} (μs)	241.28	283.81	469.17	591.99	348.17	569.58	358.59	492.15	398.73	622.95
RCV Errors	3087	8017	3119	4245	3348	5366	3798	5586	3986	6830
Call-setup Drops	2682	7002	2736	3734	2879	4646	3205	4781	3228	5624
Call-setup Messages	320141	635594	317922	315676	321645	316459	324962	318851	334935	326550
Total Messages	368496	727720	362463	358886	373972	365455	385030	372492	413491	396564
PDR	0.0084	**0.011**	0.0086	**0.0118**	0.0089	**0.0147**	0.0098	**0.0149**	0.0096	**0.0172**

TABLE III
DROP RATE AND KERNEL TIME (K_{rcv}) COMPARISON, SPS ON 2-CORE

Server Threads and CPS	Drop Rate comparison			Kernel time, K_{rcv}, comparison (μs)		
	Baseline Mode	Enhanced Mode	$\%Lower$	Baseline Mode	Enhanced Mode	$\%Lower$
2 Threads, 3200cps	0.0130	0.0060	53.84%	684.87	186.83	72.72%
4 Threads, 3800cps	0.0119	0.0082	31.09%	373.72	253.45	32.18 %
6 Threads, 3600cps	0.0113	0.0058	48.67%	436.66	264.13	39.51 %
8 Threads, 3200cps	0.0137	0.0098	28.47%	412.81	358.59	13.13%
16 Threads, 2600cps	0.0100	0.0096	4%	395.01	388.73	1.6%

TABLE IV
MEASURED SPS PERFORMANCE, BASELINE MULTI-CORE SERVER MODE, SPS ON 4-CORE

Model Parameters	Number of Server Threads									
	2 Threads		**4 Threads**		**6 Threads**		**8 Threads**		**16 Threads**	
	4000cps	4200cps	4800cps	5000cps	5000cps	5200cps	4200cps	4400cps	3800cps	4000cps
Arrival rate (packets/sec)	24000	25200	28800	30000	30000	31200	25200	26400	22800	24000
T_{sip} (μs)	63.25	61.06	86.88	79.61	94.18	94.86	120.77	121.99	177.37	188.18
K_{rcv} (μs)	427.16	518.30	220.13	212.46	165.59	192.17	212.6	221.57	209.9	259.36
RCV Errors	3318	4877	3387	3962	3276	3617	3145	3748	3101	3769
Call-setup Drops	2891	4267	3000	3542	2910	3214	2750	3305	2684	3279
Call-setup Messages	319102	315952	315704	313908	315739	315209	318907	316519	320946	318663
Total Messages	366171	361079	356395	351104	355482	354681	364633	358892	370774	366263
PDR	0.0091	**0.0135**	0.0095	**0.0112**	0.0092	**0.0102**	0.0086	**0.0104**	0.0084	**0.0103**

TABLE V
MEASURED SPS PERFORMANCE, ENHANCED MULTI-CORE SERVER MODE, SPS ON 4-CORE

Model Parameters	Number of Server Threads									
	2 Threads		**4 Threads**		**6 Threads**		**8 Threads**		**16 Threads**	
	4200cps	4400cps	5000cps	5200cps	5400cps	5600cps	4600cps	4800cps	4600cps	4800cps
Arrival rate (packets/sec)	25200	26400	30000	31200	32400	33600	27600	28800	27600	28800
T_{sip} (μs)	61.03	60.49	80.74	79.59	89.59	96.5	114.33	109.6	184.16	203.75
K_{rcv} (μs)	386.12	402.52	181.61	221.73	125.61	272.06	193.29	206.95	255.31	395.69
RCV Errors	3339	5001	3411	4642	2365	5580	3156	4215	2328	4916
Call-setup Drops	2918	4400	3038	4124	2098	4987	2789	3745	2044	4342
Call-setup Messages	318842	315232	315041	313723	316880	312176	316979	314595	318870	313507
Total Messages	364836	358278	353704	353137	357111	349292	358660	354077	363074	354953
PDR	0.0092	**0.0139**	0.0096	**0.013**	0.0066	**0.0159**	0.0088	**0.0119**	0.0064	**0.0138**

TABLE VI
DROP RATE AND KERNEL TIME (K_{rcv}) COMPARISON, SPS ON 4-CORE

Server Threads and CPS	Drop Rate comparison			Kernel time, K_{rcv}, comparison (μs)		
	Baseline Mode	Enhanced Mode	$\%Lower$	Baseline Mode	Enhanced Mode	$\%Lower$
2 Threads, 4200cps	0.0135	0.0092	31.85%	518.3	386.12	25.5%
4 Threads, 5000cps	0.0112	0.0096	14.28%	212.46	181.61	14.5%
6 Threads, 5200cps	0.0102	0.0059	42.16%	192.17	110.16	42.67%
8 Threads, 4400cps	0.0104	0.0057	45.19%	221.57	132.69	40.11%
16 Threads, 4000cps	0.0103	0.0063	38.83%	259.36	252.64	2.6%

TABLE VII
MEASURED SPS PERFORMANCE, BASELINE MULTI-CORE SERVER MODE, SPS ON 6-CORE

Model Parameters	Number of Server Threads							
	4 Threads		**6 Threads**		**8 Threads**		**16 Threads**	
	5800cps	6000cps	6000cps	6200cps	6000cps	6200cps	3800cps	4000cps
Arrival rate (packets/sec)	34800	36000	36000	37200	36000	37200	22800	24000
T_{sip} (μs)	71.78	72.12	83.03	93.07	100.29	110.13	181.02	193.93
K_{rcv} (μs)	161.10	207.9	124.3	277.4	165.35	214.04	187.3	278.03
RCV Errors	2832	4366	2489	6235	3295	5204	2640	4673
Call-setup Drops	2505	3853	2200	5408	2852	4650	2283	4058
Call-setup Messages	317172	315447	317769	316568	320129	312013	321594	317896
Total Messages	358497	357417	359417	364948	369834	349180	371789	366019
PDR	0.0079	**0.012**	0.0069	**0.017**	0.0089	**0.0149**	0.0071	**0.0127**

TABLE VIII
MEASURED SPS PERFORMANCE, ENHANCED MULTI-CORE SERVER MODE, SPS ON 6-CORE

Model Parameters	Number of Server Threads							
	4 Threads		**6 Threads**		**8 Threads**		**16 Threads**	
	6000cps	6200cps	6600cps	6800cps	6400cps	6600cps	4200cps	4400cps
Arrival rate (packets/sec)	36000	37200	39600	40800	38400	39600	25200	26400
T_{sip} (μs)	72.25	73.59	91.09	92.26	111.71	112.86	195.16	211.05
K_{rcv} (μs)	182.19	221.16	180.26	215.09	224.81	278.99	245.88	386.42
RCV Errors	2946	4586	2777	4435	3417	5509	3406	5656
Call-setup Drops	2573	3994	2344	3759	2897	4698	2978	4949
Call-setup Messages	319534	317889	325325	322247	321044	320477	318556	314689
Total Messages	365891	364950	385418	380141	378616	375724	364293	359605
PDR	0.0080	**0.013**	0.0072	**0.0117**	0.0090	**0.0146**	0.00935	**0.0157**

TABLE IX
DROP RATE AND KERNEL TIME (K_{rcv}) COMPARISON, SPS ON 6-CORE

Server Threads and CPS	Drop Rate comparison			Kernel time, K_{rcv}, comparison (μs)		
	Baseline Mode	Enhanced Mode	$\%Lower$	Baseline Mode	Enhanced Mode	$\%Lower$
4 Threads, 6000cps	0.012	0.008	33.33 %	207.9	182.19	12.36%
6 Threads, 6200cps	0.017	0.0071	58.23%	277.4	151.4	45.4%
8 Threads, 6200cps	0.0149	0.0086	42.28%	214.04	177.89	16.88 %
16 Threads, 4000cps	0.0127	0.0081	36.22%	278.03	193.68	30.33 %