

A Framework for Efficient Class-based Sampling

Mohit Saxena and Ramana Rao Kompella

Department of Computer Science

Purdue University

West Lafayette, IN, 47907

Email: {msaxena,kompella}@cs.purdue.edu

Abstract—With an increasing requirement for network monitoring tools to classify traffic and track security threats, newer and efficient ways are needed for collecting traffic statistics and monitoring of network flows. However, traditional solutions based on random packet sampling treat all flows as equal and therefore, do not provide the flexibility required for these applications. In this paper, we propose a novel architecture called CLAMP that provides an efficient framework to implement size-based sampling. At the heart of CLAMP is a novel data structure called Composite Bloom filter (CBF) that consists of a set of Bloom filters that work together to encapsulate various class definitions. In comparison to previous approaches that implement simple size-based sampling, our architecture requires substantially lower memory (upto 80x) and results in higher flow coverage (upto 8x more flows) under specific configurations.

I. INTRODUCTION

Flow monitoring is an essential ingredient of network management. Typical flow monitoring involves collection of flow records at various intermediate network boxes such as routers. These flow records can assist a network operator in various tasks such as billing and accounting, network capacity planning, traffic matrix estimation, and detecting the presence of adversarial traffic (*e.g.*, worms, DoS attacks).

While the basic task of flow monitoring appears simple, collecting flow records at high speeds under extremely resource-constrained environments is quite challenging. Particularly, memory and CPU resources in routers are often distributed among several critical functions such as route computation, forwarding, scheduling, protocol processing and so on, with the result that flow monitoring tasks receive only a small fraction of the overall pie. A classic way to overcome this hurdle is to record a random subset of packets by *sampling* the packets that traverse the interface. The rate at which packets are sampled typically depends on the resources (CPU, memory and flow export bandwidth) available on the router. The most commonly used flow collection tool today called NetFlow [3] uses a simple stage of random packet sampling.

One major deficiency of uniform packet sampling in collecting flow records is its bias towards heavy-hitter flows, *i.e.*, flows that have a large number of packets, due to the heavy-tailed flow-size distribution in the Internet. While such a bias does not affect volume estimation applications, it provides no flexibility to network operators to specify how to allocate their overall sampling budget among different classes of traffic. For example, an operator might want to specify that he is interested in collecting as many small-sized flows

as possible to satisfy security applications such as tracking botnets, detecting portscans and so on. For such applications, packet sampling is exactly the wrong choice as it inherently fills up the sampling budget with a large number of packets from “elephant” flows.

In this paper, we propose an architecture, called CLAMP, that provides network operators to perform size-based sampling. Our architecture achieves dynamic class-based sampling with the help of a novel classification data structure called Composite Bloom Filters to help network operators flexibly allocate different budgets among competing classes. Using both theoretical as well as empirical analysis, we show that our architecture achieves high flow coverage (up to 8x more flows) with substantially lower memory requirements compared to other prior approaches.

II. BACKGROUND AND RELATED WORK

The increasing importance of flow measurement as an essential ingredient in several network management tasks prompted router vendors such as Cisco and Juniper to support a simple flow measurement solution called NetFlow [3] in routers. Since observing all packets is not scalable for backbone links, routers support sampled NetFlow, a variant that works on packets that are sampled according to a configurable sampling rate (say 1 in 64). By randomly sampling packets, sampled NetFlow allows unbiased estimators to compute volume estimates for different types of traffic. Several flow monitoring solutions that exist in the literature (*e.g.*, adaptive NetFlow [4], FlowSlices [7]) are fundamentally based on this simple idea.

A recent paper by Kumar *et. al* [8], addresses the main limitation of random packet sampling—its bias towards heavy-hitters—and provides a way to perform size-dependent sampling using a sketch to estimate the flow size. FlexSample [10] builds upon this basic idea and attempts to explicitly improve the flow coverage, as opposed to just accuracy of volume estimates. Both these solutions, however, use sketches or counting Bloom filters which are quite heavy-weight in their memory requirements. In addition, they are too attuned towards size-dependent sampling and do not provide explicit ways to specify how to maximize the flow coverage or accuracy of a particular group of flows. Our architecture, on the other hand, is much more generic and also uses light-weight data structures for classification, thus reducing the overall memory consumption as well as improving the flow

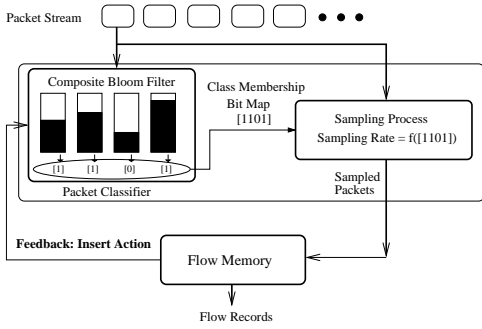


Fig. 1. Architecture of CLAMP.

coverage or accuracy depending on the specific objective, as we discuss next.

III. DESIGN OF CLAMP

Our goal is to design an architecture that provides flexibility in configuring different sampling rates for different classes of flows. The design of our architecture, CLAMP (short for CLass-based sAMpling), comprises of two basic components—a classification data structure called Composite Bloom Filter (CBF) and flow memory as shown in Figure 1. Each packet upon its arrival is passed through the CBF to first identify the class to which the packet belongs to. CBF is composed of a few Bloom filters (BFs) working in tandem to provide hints about the class to which an incoming packet belongs to. Since the CBF uses simple BFs, it is small enough to fit well in fast memory (SRAM) and can easily operate at link speeds such as OC-192 and OC-768.

Once the class to which a packet belongs to is identified, a sampling rate corresponding to this class is obtained and the packet is then probabilistically sampled based on this rate. If the packet is sampled and a flow record exists in the flow memory, then the flow record is updated with the contents of the packet (*e.g.*, packet counter is incremented, byte counter is incremented by the packet size, and special flags in the packet are noted). If not, a new flow record is created for this packet. We envision that the flow memory resides in the larger and less expensive memory (DRAM)¹. Flow memory is periodically reset when the flow records are reported to the collection center.

We apply the above framework to implement two-class size-based sampling. In particular, we consider two classes of flows—‘mouse’ and ‘elephant’ flows—which are defined as flows of size smaller (or greater) than a threshold T . The most important step is to decide which flows are inserted into a given BF. If we know the type of a flow *a priori*, then it would be straightforward to make an entry of the flow into the corresponding BF. However, all flows begin as mouse flows and only when the number of packets observed crosses a threshold that we know that a flow becomes a large flow. Thus, ideally, we need a data structure that keeps track

¹In some cases, a limited amount of flow memory may reside in SRAM which is then flushed periodically to the DRAM

of flow sizes in a scalable fashion in high speed memory. Given maintaining accurate flow sizes is hard, we can resort to approximate data structures such as counting Bloom Filters (as used by FlexSample [10]). However, counting Bloom Filters require counters which increases the size of the data structure significantly.

To reduce the memory usage, we leverage the fact that sampled packets anyway require updating the flow records in the flow memory. During that step, we can quickly check whether the flow has transitioned from a small flow to an elephant flow and then make a corresponding entry in the BF corresponding to elephant flows as depicted using a feedback action in Figure 1. One problem with this approach is that, because records in the flow memory are based on sampled packets, they are subject to re-normalization errors. When flows are inserted or transitioned between different classes based on these sampled statistics, these re-normalization errors can effect the accuracy of the mapping between a flow and a class and thus may result in mis-classification of certain flows. In addition, there could be misclassification due to the inherent false positives associated with BFs. To some extent, this is unavoidable due to the approximate nature of the data structure; the key is to ensure that such misclassification results do not significantly affect the overall flow monitoring objectives.

A. Analysis

For our example of a two-class CBF, packets can be classified as either mouse or elephant packets, with some mouse packets misclassified as elephant packets and vice-versa. We use M_M and M_E to denote sampled mouse packets that are classified as mouse and elephant packets respectively (and hence sampled at mouse and elephant sampling rates). Similarly, we denote E_M and E_E as the elephant packets classified as mouse and elephant packets respectively.

For any sampling framework, the most fundamental constraint is applied by the processing limits. We define c as the capacity or the maximum number of packets which can be sampled at a given rate constrained by the processing power.

$$N = M_S + E_S \leq c$$

where N is the total number of packets sampled and M_S and E_S are the actual number of packets sampled which belong to mouse and elephant flows respectively assuming an oracle which can perform perfect classification. From our definitions, $M_S = M_M + M_E$ and $E_S = E_E + E_M$.

Due to the presence of false positives in BFs, some packets belonging to mouse flows will be matched in the elephant BF (M_E) with a probability of $\beta = (1 - e^{kn/m})^k$, where k is the number of hash functions, n is the number of elements supported by the filter, and m is the size of the filter [6]. Hence, we need to reduce this fraction from M_M and add it to M_E . Thus, we get the following:

$$\begin{aligned} M_M &= s_M \cdot M \cdot (1 - \beta) \\ M_E &= s_E \cdot M \cdot \beta \end{aligned} \quad (1)$$

where, s_M and s_E are the sampling rates of the mouse and elephant classes and M is the total number of mouse packets. Generally, at larger sizes for the filters, β will be small enough to approximate M_M to be simply equal to $s_M \cdot M$.

Similarly, every elephant flow in the beginning of its evolution is treated as a mouse flow, until an estimated T/s_M packets have been encountered for it, where T is the threshold number of sampled packets at which a flow changes from mouse to elephant.

$$\begin{aligned} E_M &\leq F_E \cdot T \\ E_E &= s_E \cdot (E - E_M/s_M) \end{aligned} \quad (2)$$

where F_E is the total number of elephant flows and E is the total number of elephant packets. Note that E_M is almost F_E times the threshold T , since a flow will be immediately labeled as an elephant once T packets are sampled for that flow. Hence, we need to reduce these many packets sampled at mouse sampling rate, to get the expression of E_E . We now plug in the values for M_M , M_E and E_E to derive a more general inequality (assuming negligible values for β), as follows:

$$N \leq s_M \cdot M + F_E \cdot T + s_E \cdot (E - E_M/s_M) \leq c \quad (3)$$

Maximizing flow coverage: One of the main objectives we consider is increasing the number of unique flows captured by CLAMP, either for a particular group or for all the flows. To increase the flow coverage of mouse flows, we need to increase the sampling rate s_M . However, for a given sampling budget and processing constraints, we can not increase it indefinitely. Equation 3 is a quadratic in s_M which can be solved to obtain the following solution for s_M .

$$\begin{aligned} s_M &= (t + \sqrt{t^2 + 4s_E \cdot E_M \cdot M}) / (2 \cdot M) \\ \text{where, } t &= c - s_E \cdot E - E_M \end{aligned} \quad (4)$$

All positive values of s_M less than the one defined above will satisfy the processing capacity c . We can get estimates for E , M and E_M historically, based on the past measurement cycle(s). Due to the heavy-tailed nature of Internet traffic, maximizing the value of s_M will automatically lead to increasing the overall flow coverage. Thus, we maximize the value of s_M with respect to s_E to obtain the values for s_M and s_E that will lead to maximizing the flow coverage.

$$\begin{aligned} s_M &= \min\{(c - E_M)/M, 1\} \\ s_E &= \max\{0, (c - M - E_M)/(E - E_M)\} \end{aligned} \quad (5)$$

In most cases, the first configuration with s_M set to $(c - E_M)/M$ and s_E to 0 would work fine because sampling capacity c is typically very small. Only when we have higher sampling capacity, we will need to configure s_M to 1 and s_E to a value less than or equal to $(c - M - E_M)/(E - E_M)$, so as not to underutilize the spare sampling capacity. We refer to this sampling scheme as *sample and block* as it is the opposite of *sample and hold* [5] that is designed to identify large flows.

We compare the coverage gains of sample and block with that of random packet sampling with probability s_R , under the invariant of sampling budget c in both schemes.

$$\begin{aligned} G_{max} &= (s_M \cdot M) / (s_R \cdot M) = \frac{(c - E_M)/M}{c/(M + E)} \\ &= (1 + E/M) \cdot (1 - E_M/c) \\ &\text{for } E_M < c \leq (M + E_M) \end{aligned} \quad (6)$$

While G_{max} is actually packet coverage gain, it is directly related to the flow coverage gain for mouse flows, because all the mouse flows have less than or equal to T packets. G_{max} is dependent on two terms: $1 + E/M$ and $1 - E_M/c$. The first term is solely dependent on the traffic mix or the trace characteristics. However, the second term is dependent on the amount of misclassification occurring for elephant flows. In summary, sample and block samples packets belonging to mouse flows at the maximum possible sampling rate s_M , and as soon as a mouse flow becomes an elephant, it stops sampling further packets for that flow, thus increasing the mouse flow coverage.

Along with increasing the packet and flow coverage, sample and block can also increase the accuracy of the flow size estimates for the mouse flows. As we reduce s_M from its value for maximum coverage, and increase s_E to compensate for the reduction in N , accuracy in flow size estimates for medium and large flows increase. Thus, the network operator has the flexibility to configure CLAMP for achieving maximal flow coverage for mouse flows, along with sufficiently accurate flow size estimates for medium and large flows. This can be attained by tuning to a point between the configurations for maximum coverage and maximum accuracy.

IV. EVALUATION

While we envision CLAMP to be implemented in hardware for high speeds, we built a prototype software implementation of CLAMP for evaluation. This prototype required nearly 1200 lines of C++ code. The most important component of CLAMP is the packet classification data structure, which is implemented as a vector of binary Bloom filters (hash tables). Our prototype allows selecting the number of hash functions (k), number of entries (m) in each filter and even the epoch size (e). We use Bob Hash function as suggested by [9] for packet sampling at line speeds. Each hash function is initialized with a different 32 bit value.

Using this prototype implementation, we evaluate the efficacy of CLAMP over real-world traces. We also implemented other size-based sampling frameworks [8], [10] and use these in our comparisons. We show how to configure CLAMP to obtain better flow coverage using our theoretical analysis in Section III-A. We now discuss how CLAMP can be configured for maximizing the mouse flow coverage and accuracy by employing the *sample and block* scheme.

We use two real-world anonymized traces to analyze the performance of CLAMP. The first is a 10 minute OC-48 trace published by NLANR [2] with 34 million packets and about 2.2 million flows, while the second is a 1 minute OC-192

Name	Date/Time	Duration	Online Source	Link	Mbps (Kpkt/s)	Packets	5-tuple flows
ABIL	2002-08-14/09:20	600s	www.nlanr.net	OC-48	294.2 (57.6)	34,573,317	2,195,366
CHIC	2008-04-30/13:10	60s	www.caida.org	OC-192	971.4 (217.4)	13,046,322	1,174,965

TABLE I
TRACES USED FOR OUR MEASUREMENTS.

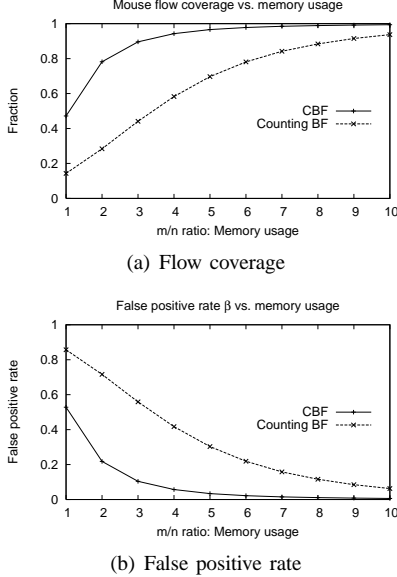


Fig. 2. Effect of varying memory usage for CBF and counting BF ($T=1, s_M=1, \text{epoch}=600\text{s}$).

backbone trace of a tier-one ISP published by CAIDA [1] with 12 million packets and 1.1 million flows. Both traces exhibit heavy-tailed flow size distributions that we assume in our analysis. These traces are summarized in Table I.

A. Memory Usage

In this section, we compare CLAMP with other counter-based schemes for size-based sampling such as FlexSample [10] and sketch-guided sampling [8]. While the focus and usage of these solutions is different, they both share similar data structures (sketch and counting Bloom filters) to keep track of the flow sizes. In contrast, we rely on CBF that represents a much simpler data structure. In many cases, such simplicity comes at the cost of worsening some other metric such as, say, increasing the false positive rate of the filter. The results indicate, somewhat surprisingly, that CBF performs better in both memory consumption as well as false positives compared to counting BF alternatives, thus indicating that CBF achieves clear benefit and is not a tradeoff.

Why is higher false positive rate in the classification data structure bad for various sampling objectives? As an example, consider the case when we are interested in increasing flow coverage. According to Equation 1, the mouse flow coverage is directly related to the number of mouse packets classified and sampled at mouse rate M_M . Mouse flow coverage, however, decreases as filter false positive rate β increases as these packets will be misclassified as elephant packets and

thereby sampled at elephant rate (*i.e.*, less than the mouse rate for high flow coverage). Thus, low false positive rate for the classification data structure is important for such objectives.

In Figure 2(a), we compare the mouse flow coverage (which is equal to M_M for T set to 1) for the full ABIL trace for CLAMP and counting BF. In addition to the mouse flow coverage, we also plot the empirical values for filter false positive rate β (calculated using Equation 1) in Figure 2(b). The configuration allowed mouse flow coverage to be maximum, *i.e.*, s_M set to 1, thus ensuring a worst case analysis for the filter with the maximum number of flows inserted in the Bloom filter over the trace. To ensure fair comparison, we use three hash functions for both CBF as well as the counting Bloom filter.

Figures 2(a) and 2(b) show that CLAMP as well as counting BF implementations exhibit a reduction in the false positives and increase in the flow coverage as we increase the amount of memory over-provisioning (m/n). However, clearly, CLAMP exhibits much faster increase in mouse coverage and decrease in false positives (or mis-classifications). CBF achieves a filter false positive rate of 6% at a m/n ratio of 4 (using 174.5KBytes of memory), giving nearly 94.3% mouse flow coverage (at $s_M=1$ and $s_R=0.069$). Note that we do not have 100% mouse flow coverage due to the fact that M_E is not zero. At the same sampling rate, the counting Bloom filter implementation requires a m/n ratio of 10 (using 13,960 KBytes of memory) to achieve the same filter false positive rate of 6%. This shows that CBF requires nearly 80x less high-speed SRAM than a counting Bloom filter to achieve similar filter false positive rates and flow coverage.

On the other hand, even considering only same number of entries (bits and counters) across both CBF and counting Bloom filter, CBF still obtains at least up to 6x reduction in the number of false positives (at $m/n = 5$, CLAMP has about 4% false positives while counting Bloom filter experiences about 25% false positives). What makes these gains even more significant is the fact that we do not consider the extra overhead associated with counters (in the counting Bloom filters). If we factor in this disparity, the benefit associated with CBF will increase even further.

B. Flow coverage

According to Equation 5 and Equation 6, the best possible coverage for mouse flows can be achieved by setting CLAMP in *sample and block* mode with $T=1$. Table II shows the results for ABIL trace with epoch size of 600s and CBF allocated 436.25 KBytes of memory. We note that the theoretical gains for mouse coverage follow Equation 6. Those gains are formulated for the mouse flows which are all such flows with

		Flow Coverage %		
s_R	s_M	Pkt Sampling	CLAMP	Gain
0.001	0.0043	0.3	1.1	3.74x
0.004	0.0267	1.0	5.2	5.08x
0.016	0.146	3.4	20.8	6.17x
0.064	0.845	10.7	86.4	8.10x
0.073	1.0	TABLE II	99.4	8.32x

COVERAGE FOR FLOWS OF SIZE 0-1K PACKETS (ABIL TRACE, $T=1$, EPOCH=600S, MEMORY=436.25KBYTES).

		Flow Coverage %		
s_R	s_M	Pkt Sampling	CLAMP	Gain
0.001	0.0010	0.4	0.4	0.99x
0.004	0.0041	1.6	1.7	1.03x
0.016	0.0179	6.0	6.7	1.11x
0.064	0.0965	18.7	25.3	1.36x
0.307	1.0	53.8	99.2	1.84x

TABLE III
COVERAGE FOR FLOWS OF SIZE 0-100 PACKETS (CHIC TRACE, $T=5$, EPOCH=10S, MEMORY=50KBYTES).

size less than or equal to T packets ($T=1$). However, in Table II we show the gains for flows of size 0-1000 packets, which will be slightly less than those obtained for mouse flows according to Equation 6. The flow coverage gains for the overall traffic volume are also mainly decided by the mouse flow coverage, because large flows are almost always captured.

We can observe from Table II that as we increase the random sampling probability s_R , the coverage gain of CLAMP increases from 3.74x (at $s_R = 0.001$) to almost 8.32x (at $s_R = 0.073$). Further increasing s_R will reduce the gain since at $s_R = 0.073$, the mouse sampling rate s_M is already set to 1 and it cannot further increase the flow coverage. As mentioned earlier, even with $s_M = 1$, we can see that CLAMP almost captures 99.4% of traffic (with the remaining 0.6% attributed to the false positive probability associated with the elephant Bloom filter). Thus, we can clearly conclude that CLAMP provides an order of magnitude better coverage for this traffic mix as compared to random packet sampling.

For the CHIC trace, we show the results in Table III. A maximum gain of 84% is achieved for this trace at a random packet sampling rate of 0.307, while at lower sampling rates ($s_R=0.016$), the gains are just 11%. There are two important observations: First, at very low sampling rates (~ 0.001), CLAMP is almost as bad as random packet sampling since there is not enough sampling budget mouse flows can ‘steal’ from elephants to improve their coverage. Second, at high sampling rates such as 0.307, we get a gain of 84% which is not as good as we obtained for the ABIL trace (8.32x at $s_R=0.073$). This is in part because of the flow size distribution of CHIC trace (see Equation 6 for the $1 + E/M$ term). But the major reason is attributed to the high E_M/c ratios for all of these configurations in Table III, because of smaller epoch size (10s) and higher threshold ($T=5$), effectively decreasing the coverage gain (see Equation 6 for the $1 - E_M/c$ term). The role of E_M/c ratio is further strengthened by the fact that operating in *sample and block* mode for the CHIC trace

(with $T=1$, epoch=60s), a lower sampling rate ($s_R=0.138$) results in a much better coverage gain of 3.05x (with only flow memory reset after every 10s), thereby reducing the share of E_M/c . This shows that CLAMP can nevertheless achieve an order of magnitude better flow coverage than random packet sampling even for CHIC trace, with suitable tuning, based on Equation 6. For brevity, we omit those results for CHIC trace.

V. CONCLUSIONS

Flow monitoring solutions in routers have evolved significantly over the years from their modest origins in simple NetFlow-like solutions. While most solutions revolve around better handling router resource constraints such as CPU, memory and flow reporting, there is little research on providing an efficient and flexible class-based sampling architecture, with dynamic class definitions that include specific flow properties such as the size. In this paper, we have discussed the architecture of CLAMP to address this challenge that involves the use of a set of simple Bloom filters for class membership and a feedback from the actual flow memory to record class-membership information about flows. Using this architecture, we have shown how to implement a simple two-class size based packet sampling framework. Compared to prior approaches, we achieve significant memory reduction (up to 80x depending on the configuration) and increased number of flows (up to 8x). There are several directions for future work. For example, we can extend our analysis to multiple classes. We can also consider applying our architecture to other definitions of classes. We plan to pursue these in our future work.

VI. ACKNOWLEDGEMENTS

This work was supported in part by NSF Award CNS 08316547 and a grant from Cisco Systems.

REFERENCES

- [1] CAIDA Anonymized 2008 Internet Trace (equinix-chicago collection). http://www.caida.org/data/passive/passive_2008_dataset.xml.
- [2] NLNR Abilene-I Internet dataset. <http://pma.nlanr.net/Traces/long/ipls1.html>.
- [3] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954.
- [4] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *Proc. of ACM SIGCOMM*, 2004.
- [5] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *SIGCOMM*, 2002.
- [6] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.
- [7] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *Proc. of IMC*, 2005.
- [8] A. Kumar and J. J. Xu. Sketch guided sampling - using on-line estimates of flow size for adaptive data collection. In *IEEE INFOCOM*, 2006.
- [9] M. Molina, S. Niccolini, and N. Duffield. A comparative experimental study of hash functions applied to packet sampling. In *Technical Report, AT&T*.
- [10] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Building a Better Mousetrap. In *Georgia Tech CSS Technical Report GIT-CSS-07-01*, 2007.