# String Matching Algorithms for High-Speed Network Content Security Processing: Postprint

**Authors:** He Huimin, Liu Xia, Jiang Lei

**Date:** 2017-03-10T00:00:00+00:00

## Abstract

The proliferation and development of Internet technologies have exacerbated the cybersecurity landscape, imposing higher demands on network content security processing technologies. As the core technology for network content security processing, pattern string matching algorithms are also facing new challenges. To address the requirements of high-speed network content security processing, this paper proposes three categories of string matching algorithms: software algorithms (e.g., optimal window selection algorithm), instruction set algorithms (e.g., SSE instruction set-based string matching algorithm), and hardware algorithms (e.g., FPGA-based string matching algorithm). These algorithms substantially enhance the speed of pattern string matching, satisfying the requirements of high-speed network content security processing systems.

## Full Text

### String Matching Algorithms for High-Speed Network Content Security Processing

**He Huimin, Liu Xia, Jiang Lei**

## Abstract

The proliferation and development of Internet technologies have exacerbated cybersecurity threats, placing higher demands on network content security processing technologies. As the core technology for network content security processing, pattern string matching algorithms face new challenges. This paper proposes three types of string matching algorithms tailored for high-speed network content security processing: software algorithms (e.g., optimal window selection algorithm), instruction set algorithms (e.g., SSE instruction set-based string matching algorithm), and hardware algorithms (e.g., FPGA-based string matching algorithm). These algorithms significantly improve pattern string

matching speed and meet the requirements of high-speed network content security processing systems.

**Keywords:** High-speed network content security processing; Optimal window selection; Instruction set algorithms; Hardware algorithms

## 1 Background and Research Significance

In recent years, the proliferation of Internet technology has led to increasingly severe cybersecurity threats stemming from vulnerabilities in Internet protocols and computer systems. Traditional network content security processing technologies have become a bottleneck in the development of the field, as their storage space and computational speed can no longer meet the demands of real-time high-speed network environments. String matching technology for high-speed network content security processing—characterized by fast processing speeds and high real-time performance—has garnered widespread attention from both academia and industry, emerging as a hot research topic in cybersecurity.

Pattern string matching algorithms constitute the core technology of network content security processing and represent a focal point of research in computer science. In the cybersecurity domain, network content security processing technologies primarily employ pattern matching algorithms to analyze the content payload of network packets and identify/classify specific network data according to predefined rules. Typical applications include Intrusion Detection/Prevention Systems (IDS/IPS), Anti-Virus/Anti-Spam detection (AV/AS), network bandwidth management and Quality of Service (QoS), and Unified Threat Management (UTM).

Pattern string matching is defined as follows: given a pattern string set $P$, for any input text $T$, find all occurrences of patterns from $P$ in $T$. Broadly speaking, pattern string matching algorithms include exact multi-pattern string matching algorithms, regular expression matching algorithms, and approximate string matching algorithms. Exact multi-pattern string matching is the most mature technique and has been widely applied in intrusion detection systems, becoming a decisive factor affecting system performance. However, as virus rules posing network content security threats have grown increasingly complex in recent years, exact string rules sometimes cannot precisely express users' semantic requirements in application systems, necessitating the use of regular expressions to represent more complex semantics. The work in this paper addresses both exact string matching and regular expression matching.

## 2 Related Work

Pattern string matching algorithms are not only the core technology for network content security processing but also a critical factor affecting the development of text retrieval, search engines, language translation, spell checking, and biological

computing. Consequently, numerous improvements to classical string matching algorithms have emerged in recent years, with new algorithmic designs appearing continuously.

Recent research from both domestic and international scholars has produced many new results in exact string matching algorithms. Reference [1] proposed LDM (Linear DAWG Matching), a time-complexity-optimal exact string matching algorithm that divides text into overlapping windows and comprehensively uses suffix automata DAWG and AC automata for scanning within each window, guaranteeing linear worst-case time complexity and sub-linear average time complexity. References [2, 3] employed Bloom Filter technology for string matching, grouping characteristic strings by length, with strings of the same length placed in the same group and each group filtered by a Bloom filter engine.

The academic community has also conducted in-depth research on regular expression matching techniques. Reference [4] used rule classification and rule rewriting methods to simplify regular expressions, proposing two rewriting rules to reduce the number of states in linear scale. However, this approach is only suitable for non-overlapping matching scenarios, limiting its applicability. References [5, 6] proposed D2FA (Delayed input Deterministic Finite Automata) and delta-FA (Delta Finite Automata) to compress the storage space of deterministic automata. Both methods utilize relationships between adjacent states to reduce the number of transitions in transition tables, thereby reducing automata storage space, but they suffer from the drawback of time-consuming state updates, resulting in suboptimal actual matching performance.

Despite the abundance of existing string matching algorithms, several shortcomings remain: (1) these algorithms can only handle pattern string sets of medium scale or smaller, with performance degrading significantly as the pattern set size increases; (2) these algorithms are only effective for certain specific types of pattern strings and lack universality; and (3) the aforementioned hardware algorithms cannot meet the requirements of real-time network content processing in practical systems. Therefore, there remains a need to design better pattern string matching algorithms.

## 3 Our Research Work

We approached the problem from both exact string matching technology and regular expression matching technology, proposing three types of string matching algorithms: software algorithms, instruction set algorithms, and hardware algorithms. The software algorithm designs an optimal window selection algorithm; the instruction set algorithm utilizes the SSE instruction set to optimize the SOG algorithm and classical bit-parallel algorithms; and the hardware algorithm implements a regular expression matching algorithm based on Field-Programmable Gate Arrays (FPGA). These algorithms have achieved excellent results in improving matching speed.

### 3.1 Software Algorithm: Optimal Window Selection Algorithm

Suffix algorithms are widely used in string matching due to their high matching speed, but they require patterns to be of equal length—a condition that often cannot be satisfied in practical applications. General suffix algorithms employ simple processing methods, such as the Wu-Manber algorithm [7], which selects the leftmost $m$-character substring from patterns, where $m$ is the length of the shortest pattern in the set. This processing method does not consider the impact of the selected substring on matching performance. We propose a substring extraction method that extracts a partial string set from the pattern string collection to construct a basic data structure, which is then used to scan the text string to achieve maximum possible matching speed.

We adopt the SBOM (Set Backward Oracle Matching) algorithm [8] to complete the extraction of partial string sets from the pattern collection during the training process, as shown in Figure 1 [Figure 1: see original paper]. Let $T$ denote the text string, $n$ the text length, $m$ the length of the shortest pattern in the set, $k$ the total number of sliding window shifts during text searching, and $d\_i$ the number of character comparisons in the $i$-th window. The total time cost is given by:

$$\text{totalcost} = \sum_{i=1}^{k} d_i$$

and the average cost is:

$$\text{averagecost} = \frac{\text{totalcost}}{n}$$

From the formula, average cost is minimized when $k$ is minimized. While $k$ is determined by the $m$-length partial string set $Q$, the number of possible $Q$ is too large to enumerate and find the minimal $k$ value.

Therefore, we adopt an approximately optimal method that calculates the time cost for each partial string of a pattern, thereby selecting the partial string with minimum time cost. The union of all such selected partial strings yields $Q$.

The specific steps are as follows. For any partial string $q\_\{ij\}$ in pattern $p\_i$:

**Step 1:** First, compute all its substrings, then calculate each substring' s occurrence count $c\_k$ in the training text. Let the length of substring $s\_k$ be $z$, then each substring' s time cost is $c\_k$.

**Step 2:** Accumulate all substring time costs to compute each partial string' s total time cost.

**Step 3:** Repeat Steps 1 and 2 to compute time costs for other partial strings in the pattern, then select the partial string with minimum time cost.

**Step 4:** Repeat Steps 1-3 to compute optimal partial strings for other patterns.

Using pattern strings from Snort [9] as experimental data, we obtained the results shown in Figure 2 [Figure 2: see original paper]. In the figure, "right," "left," and "middle" refer to extracting $m$ characters from the rightmost, leftmost, and middle positions, respectively. "Mincost" represents substrings extracted by our algorithm. The results show that although matching speeds for all four algorithms decline as pattern set size increases, our algorithm is faster than the other three substring extraction methods—approximately 20% faster than the "right" method—and its speed decreases more slowly as pattern set size grows.

### 3.2 Instruction Set Algorithm: SSE-Based String Matching Algorithm

The SSE (Streaming SIMD Extensions) instruction set provides a group of wide registers supporting 128-bit parallel data operations. This paper utilizes the SSE instruction set to optimize classical bit-parallel algorithms (Shift-And and BNDM) and the SOG algorithm.

Shift-And and BNDM are classical bit-parallel algorithms. Let machine word length be $w$, text length be $n$, number of patterns be $r$, and shortest pattern length be $m$. The time complexity of Shift-And and BNDM algorithms is $O(n\lceil mr/w \rceil)$. Due to bit-parallel technology, Shift-And and BNDM achieve fast matching speeds. However, once the total pattern length exceeds machine word length, both algorithms' performance degrades significantly.

The basic idea of optimizing Shift-And and BNDM using SSE is: employ a bit vector $D$ to record the matching state between each pattern's prefix or substring and the current text $t$, packing multiple patterns' state vectors into 128-bit SSE registers (as shown in Figure 3 [Figure 3: see original paper]). During matching, each text character read triggers updates to state vector $D$ through SSE bit operation instructions (logical OR, logical AND, left shift), and SSE bit comparison instructions detect specific bits to determine successful matches.

In experiments comparing optimized Shift-And (called SSE-Shift-And) and optimized BNDM (SSE-BNDM) against original algorithms (Figure 4 [Figure 4: see original paper]), both SSE-Shift-And and SSE-BNDM achieve excellent acceleration, with speedup ratios remaining essentially constant as pattern count increases.

The SOG algorithm [10], proposed by L. Salmela et al., is a matching method for pattern sets of 100,000-scale. The algorithm reduces pattern set $P$ to a wildcard pattern $p$, searches for $p$ using Shift-OR, and finally uses secondary hashing and binary search to verify potential matches and report all true matches.

From SOG's basic principle, its filtering stage false positive rate is $(1-(1-1/\sigma)^q)^m$, where $\sigma$ is alphabet size and $q$ is wildcard count. As pattern set size $r = |P|$ increases, SOG's filtering effectiveness deteriorates, requiring more verification operations and significantly reducing matching speed. To address

this, we improve SOG' s filtering effectiveness through pattern set grouping and reduction.

The basic idea of pattern grouping reduction: divide pattern set $P$ into $g$ groups: $P = P_1 \cup P_2 \cup \cdots \cup P_g$. Each group $P\_j$ is independently reduced to a wildcard pattern $p_j$, then these $g$ wildcard patterns are packed into a machine word for parallel SOG searching. Let machine word width be $w$. Since each wildcard pattern $p_j$ requires $m - q + 1$ bits, the pattern set can be divided into at most $g = \lfloor w/(m - q + 1) \rfloor$ groups.

Our algorithm uses maximum bipartite matching theory to select optimal windows for patterns, forming the optimized SOGOPT algorithm. We introduce SSE instructions into the grouped reduction SOGOPT algorithm, using SSE register width to reduce verification operations. The method employs bit vector $D$ to record each wildcard string prefix' s matching state with current text $T$, packing state vectors into 128-bit SSE registers. During matching, each text character read triggers state vector $D$ updates through SSE bit operations, with specific bits detected to determine match success.

We collected approximately 100GB of URL data from backbone routers and compared SOGOPT against original SOG and multiple string matching algorithms (AC, WM, SBOM, KR, DTM) in terms of matching speed and storage space. Results show SOGOPT achieves the fastest matching speed while maintaining reasonable memory consumption, as demonstrated in Figures 5 [Figure 5: see original paper] and 6 [Figure 6: see original paper].

### 3.3 Hardware Algorithm: FPGA-Based String Matching Algorithm

With the rapid growth of network bandwidth, hardware methods for regular expression matching have attracted significant attention. Numerous hardware acceleration schemes based on Non-deterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA) have been proposed. NFA algorithms are primarily logic-circuit-based, requiring substantial logic resources, while DFA algorithms store state transition tables in memory, requiring significant memory resources. To address DFA' s high memory consumption while retaining NFA' s high-performance advantages, we propose an improved DFA matching algorithm implemented entirely in FPGA logic circuits.

Yang Yifu [11] observed that most transition edges in DFA states converge to the same state. We leverage this characteristic by merging these identical transition edges into a "default transition." The state pointed to by the default transition is called the default state. By recording only this default transition edge, we can significantly reduce the number of transition edges and simplify circuit structure.

Figure 7 [Figure 7: see original paper] shows the improved DFA matching circuit for regular expression "a(b|c)d." The circuit implements default transition edges using cascaded AND and NAND gates. Outputs from non-default states in each

state transition table connect to their respective state registers and also to a default transition circuit composed of NOR and AND gates. When the current state is active and the input character' s next state is the default state, all AND gates in that state transition table output low level, causing the NOR gate' s input to become high level, indicating the next state should be the default state. This approach eliminates numerous unnecessary logic gates and reduces circuit fan-out, improving system throughput. For the regular expression "a(b|c)d," the original DFA scheme requires 1,543 logic gates, while the improved design reduces this to only 38 gates.

To comprehensively and realistically compare our improved DFA matching engine' s performance against Sidhu' s [12] NFA matching engine implementation, we extracted 102 regular expression rules from the L7-filter rule set as test data, recording logic element (LE) consumption and clock frequency (MHz) for both NFA and DFA algorithms. Test results show that for 62% of rules, the DFA algorithm consumes fewer logic resources than NFA; for approximately 10% of rules, DFA throughput exceeds NFA; for 60% of rules, both algorithms achieve equivalent throughput; and for 30% of rules, DFA throughput is lower than NFA.

## 4 Conclusion

This paper designs three types of string matching algorithms: software algorithm (optimal window selection algorithm), instruction set algorithm (SSE instruction set-based string matching algorithm), and hardware algorithm (FPGA-based string matching algorithm). These algorithms significantly improve pattern string matching speed and are suitable for high-speed network content security processing systems. Our future work will focus on designing string matching algorithms for ultra-large-scale pattern sets and specialized system architectures for high-speed network content security processing.

## References

[1] 贺龙涛, 方滨兴, 于翔湛. 一种时间复杂度最优的精确串匹配算法. 软件学报. 2005.

[2] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. IEEE Micro, 24(1):52–61, 2004.

[3] Michael Attig, Sarang Dharmapurikar, John Lockwood. Implementation Results of Bloom Filters for String Matching. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, April 20-23, 2004.

[4] Fang Yu, Zhifeng Chen, Yanlei Diao. Fast and memory-efficient regular expression matching for deep packet inspection. Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems 2006, San Jose, California, USA December 03-05, 2006.

[5] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan Turner, George Varghese. Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems, December 03-04, 2007, Orlando, Florida, USA.

[6] Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci, Gianni Antichi, Andrea Di Pietro. An improved DFA for fast regular expression matching. ACM SIGCOMM Computer Communication Review, Volume 38, Issue 5(October 2008), Pages 29-40.

[7] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching" , Dept. of Computer Science, University of Arizona, Tucson, AZ, TR-94-17, 1994.

[8] (Citation appears incomplete in original)

[9] Snort pattern strings used as experimental data.

[10] L. Salmela et al. SOG algorithm for 100,000-scale pattern matching.

[11] Yang Yifu. Observation on DFA state transition concentration.

[12] R. Sidhu. NFA matching engine design and implementation.

*Note: Figure translations are in progress. See original paper for figures.*