Xpriori Technical Paper

NeoSlider Packet Search Engine with Optional Pre-Parser and Optional Proximity Search Engine

by Harry Direen, Ph.D.

Release 1.1

NOTE: In October 2003, Xpriori, LLC acquired NeoCore Holdings, LLC including all technology and patents. Any references to Neo, NeoCore or NeoCore Holdings, LLC technology or patents as such are now the property of Xpriori, LLC.

Xpriori, LLC
2864 S. Circle Dr.
Ste. 401
Colorado Springs, CO 80906
(719) 425-9840
www.xpriori.com

Version 1.1

Xpriori technology is protected by the following patents:
US Patent #5,742,611 (21 Apr 98)
US Patent #5,942,002 (8 Aug 99)
US Patent #6,157,617 (5 Dec 00)
US Patent #6,167,400 (26 Dec 00)
US Patent #6,324,636 (27 Nov 01)
US Patent #6,493,813 (10 Dec 02)
US Patent #6,792,428 (14 Sept 04)

Other U.S. and international patents pending.

The information in this white paper has been provided by Xpriori, LLC. To the best knowledge of Xpriori, it contains information concerning the current state of information processing technology. Xpriori, LLC disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to products described in this paper, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. No specific reliance should be made on the material provided herein without thorough investigation of the technology and its proposed application to specific circumstances. Product and technology information is subject to change without notice.

Introduction

The NeoSlider Packet Search Engine is used for full content scanning of streams of data. The stream of data may be broken up into small, variable length sets of data, as is typical with packet payloads coming in over a network. The search engine will treat the packet data sets as one long continuous stream of data. Virtually any number of items may be searched for in the stream of data at the same time without impacting the performance of the search engine. The items being searched for can be any defined byte sequence. This could be objectionable words, company project names, URLs, and so on.

An optional pre-parser may be added to the search engine. The pre-parser enables mapping any character to any character. This can be used to: map upper case letters to lower case letters (or visa versa); map various white space characters to a common white space character; and/or other mappings as required. The pre-parser also enables ignoring characters and removing duplicate characters. If all white space characters are mapped to a common white space character, all duplicates of the white space character may then be ignored in the search process.

An optional proximity search engine may be added to the packet search engine. Proximity searches are looking for something like “fire” and “smoke” within 100 characters of each other. The engine would produce a hit only if both words, “fire” and “smoke” were found, and only if they were found within 100 characters of each other. The number of items combined to form a given proximity search is not limited. We could look for “Smokey”, “bear”, “forest”, “fire” and “smoke”, all within a given distance of each other. Proximity searches add powerful search criteria to the NeoSlider Packet Search Engine.

Multiple NeoSlider Packet Search Engines may be instantiated at the same time against a common database of items being searched for. In this fashion, multiple streams of data (or multiple sessions) may be searched at the same time. Information coming in over a network arrive a packet at a time in somewhat random order, headed for a variety of applications. As packets arrive, they are sorted by the session they belong to (application they are headed for) and sorted for proper order by session. After the sorting process, packets headed for an application that requires content scanning can be passed to the NeoSlider Packet Search Engine. After content scanning, the information can be passed on to the application.

The NeoSlider Packet Search Engine is based on NeoCore’s Thee-Dimensional Content Scanning using Digital Pattern Processing (DPP™). The various technical papers noted in the appendix cover the details of this technology.

NeoSlider Packet Search Engine

Figure 1 is a block diagram of the NeoSlider Packet Search Engine. The packet search engine contains input and output queues for the stream of data, an optional pre-parser, the NeoSlider three-dimensional content search engine, and a hit queue for data items found. The database (NeoStore) of items being searched for is functionally separate from the search engine. This allows multiple search engines to be instantiated against a common NeoStore.

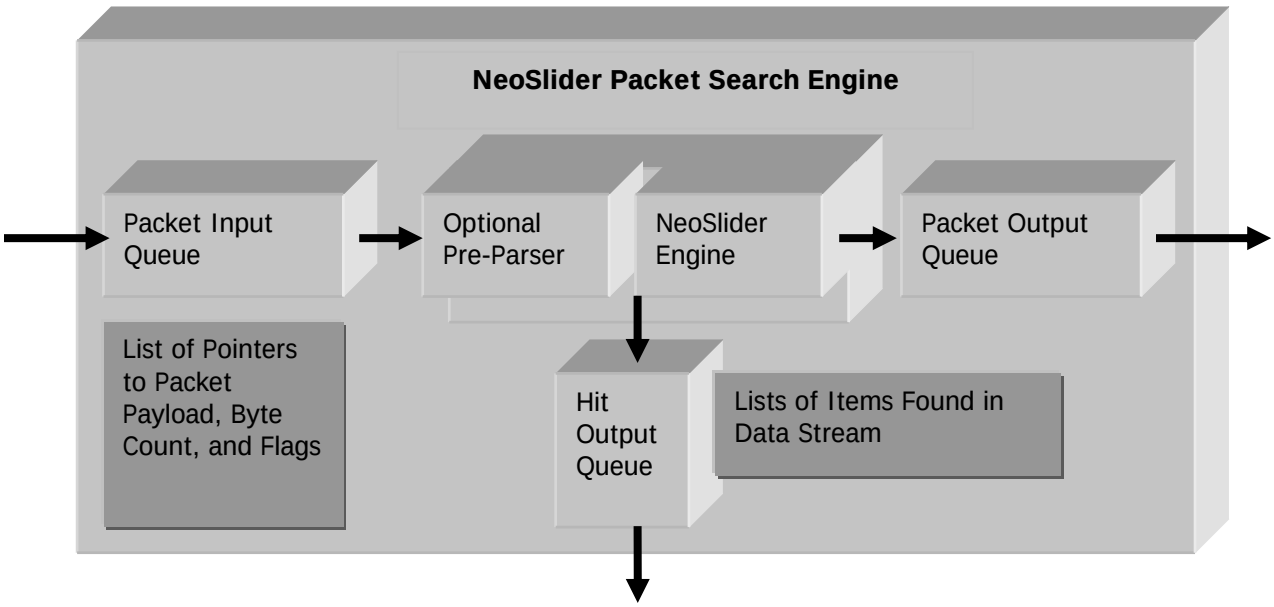

Figure 1

A packet structure (header) is established which contains: 1) a pointer to the start of a packet of data; 2) a counter containing the number of bytes in the packet; 3) start of data and end of data flags. Packets of information are passed to the search engine by filling in the packet structure and putting this structure into the input queue. The start flag indicates if the packet is the beginning of a new data stream. When the start flag is set, the search engine resets internal states and treats the data in this packet as the beginning of a new data stream. Intermediate packets will have both the start and end flags cleared. The last packet of data in a stream should have the end flag set. This tells the engine to complete the search on this packet and not to look for additional data beyond the end packet. If the entire stream of data is contained in one packet, both the start and stop flags would be set. Packets of data may be as small as one character or as large as desired.

The NeoSlider Packet Search Engine will start processing the data stream as soon as packets of data are available. Once a packet of data is completely scanned, the packet structure is moved to the output queue. The actual data being scanned is not moved in memory, only the packet header is moved to the output queue. Moving the packet header to the output queue logically informs the system that the search engine is done scanning that packet of information. This is useful when the search engine is effectively in series with a data stream being passed to another application. The output queue is optional and may be removed or ignored for applications that do not require it. If no data packets are available in the input queue, the search engine sits idle waiting for more data.

The NeoSlider search engine has an effective sliding window for the items being searched for. The last packet in the input queue cannot be moved to the output queue until either a new

packet of information comes along and the sliding window moves fully inside the new packet, or the end packet flag is set. If the end flag is set on the current packet of data, when the sliding window hits the end of the data in that packet, the search engine stops searching that data stream. The end packet will be moved to the output queue, informing the system that the search process on that data stream is complete. New data streams may be started immediately.

The search engine contains an optional, table driven, pre-parser. The pre-parser enables mapping any character to any character. This can be used to: map upper case letters to lower case letters or lower case to upper case letters; map various white space characters to a common white space character; and/or other mappings as required. The pre-parser also enables the system to ignore characters and remove duplicate characters. If all white space characters are mapped to a common white space character, all duplicates of the white space character may then be ignored in the search process. For example, suppose we are searching for "John Smith" in a stream of data. The problem is that "John" could be at the end of one line and "Smith" could be at the beginning of the next line. There will be a line-feed, carriage return, and possibly multiple spaces between "John" and "Smith": "John<CR><LF><Sp><Sp>Smith". To make matters worse, word processors tend to place a variety of non-visible characters in a file. By mapping all white space characters to say <Sp> and ignoring multiple white spaces, and ignoring non-printable characters, "John<CR><LF><NonPrintableChars><Sp><Sp>Smith" will be mapped to "John<Sp>Smith" which would then be found. The pre-parser does not change or affect the actual data stream, it only changes what the NeoSlider engine sees and searches against.

The NeoSlider engine performs content scanning against a virtually infinite list of items (keys) being searched for. The items being searched for are contained in a separate database called the NeoStore. A record is made of all items found in the data stream and the record is stored in the Hit Output Queue. Actions taken based on Hits are end use dependent. The search engine can also optionally be stopped whenever an item in the database is found in the data stream. The engine will then wait for the system to take action on the item found.

When an item is found in the data stream a variety of information can be captured at that time. The information captured will depend on the system requirements. Information that may be captured is:

- Offset into the data stream
- Offset into the packet that contains the start of the data item
- Packet that the item was found in
- Reference to the item found
- Other information as required by end user

A structure is defined to store the "hit" information in. As noted above, when a hit occurs the required information is copied to the structure and then passed either to the Hit Queue or directly to the application. If the hit information is passed to the application the scan engine stops until the application processes the hit and restarts the engine.

Proximity Search Engine

Figure 2 shows the NeoSlider Packet Search Engine with the optional Proximity Search Engine added. As noted above, proximity searches are looking for something like "fire" and "smoke" within, say, 100 characters of each other. The engine would produce a hit only if both words, "fire" and "smoke" were found, and only if they were found within 100 characters of each other. The number of items combined to form a given proximity search is not limited. We could look for "Smokey", "bear", "forest", "fire" and "smoke", all within a given distance of each other.

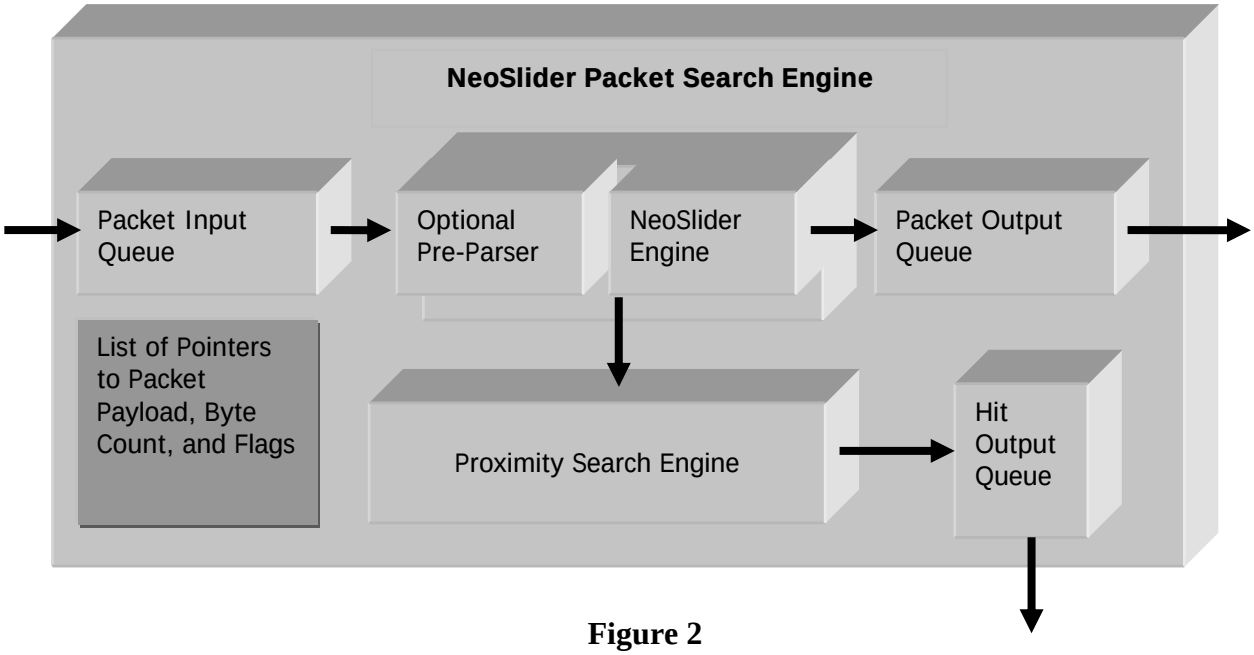

Figure 2

The hit outputs of the NeoSlider Packet Search Engine feed the Proximity Search Engine. Only the combined items, found by the NeoSlider engine, that meet all the qualifications of the proximity search, reach the final Hit Output Queue and are reported to the end application. The results in the final Hit Output Queue can be as simple as an indication that the given proximity list was found, or as complex as the full details as to where each item in the proximity list was found in the data stream.

When the proximity engine is used, it is still possible to search for single key items and, at the same time search for proximity lists of items.

Multiple Search Engines

A NeoSlider Packet Search Engine is a software object. Multiple engines can be instantiated to scan multiple data streams or follow multiple network sessions. This case is depicted in figure 3. The list of items being searched for is contained in the NeoStore and all search engines may work against the same store. The different search engines can be handled via polling methods or by placing them on separate operating threads.

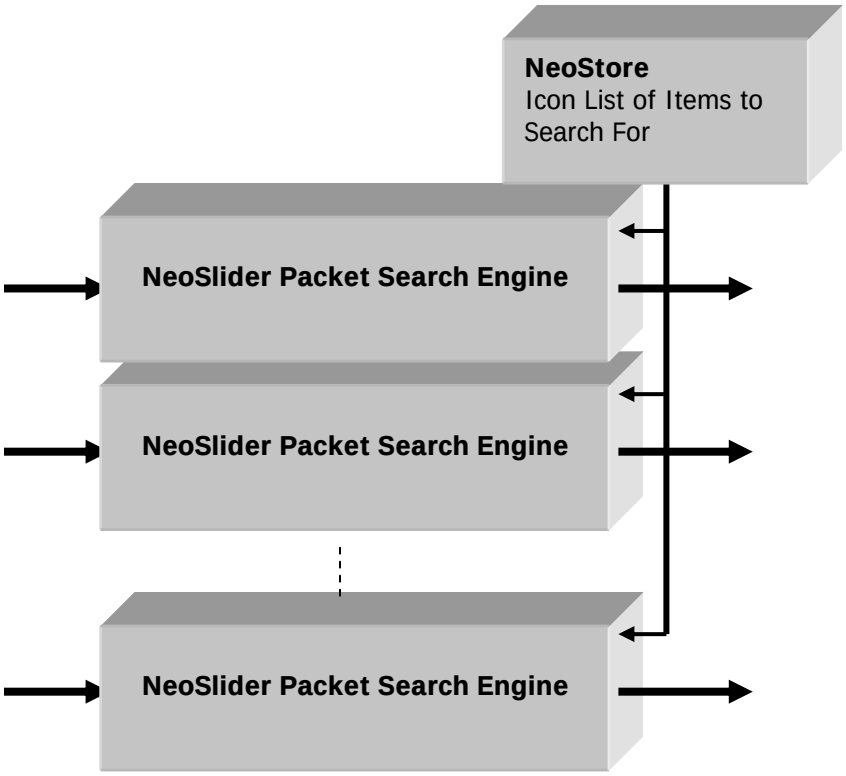

Figure 3

The NeoStore is the database of items being searched for. The NeoStore is a virtual CAM (Content Addressable Memory). Methods are available for adding and removing items from the NeoStore.

Conclusion

The NeoSlider Packet Search Engine provides fast, efficient, full content scanning of data streams. Virtually any number of items may be searched for in the stream of data at the same time without impacting the performance of the search engine. The stream of data may be broken up into small, variable length sets of data, as in packet payloads coming in over a network. The search engine will treat the packet data sets as one long continuous stream of data. The search engine contains an optional pre-parser and an optional Proximity Search Engine to aid in powerful data searches. By instantiating multiple search engines, multiple streams of data may be searched at the same time.

Appendix

Related Papers

Brandin, Chris, “A Definition of Digital pattern Processing™”

Brandin, Chris, “DPP™ Memory Management”

Brandin, Chris, “Three-Dimensional Content Scanning Using DPP™”

Brandin, Chris, “Optimized Coding Methods for Icon Generation and Manipulation in DPP™”

Brandin, Chris, “Behavioral Set and Field Descriptor Implementations in DPP™”

Brandin, Chris, “Management of Duplicate Data Elements in DPP™ Virtual Associative Memories”

Direen, Harry, “Duplicate Tree Structures in DPP™ Virtual Associative Memories”

Direen, Harry and Phillips, Keith, “Finite Fields and Properties of the Xpriori Icon Generator, Associative Processing Unit, and Associative Memory Controller used in Digital Pattern Processing™”

# # #