SiLK: A Tool Suite for Unsampled Network Flow Analysis at Scale

Mark Thomas, Leigh Metcalf, Jonathan Spring, Paul Krystosek, Katherine Prevost
netsa-contact@cert.org
CERT® Coordination Center, Software Engineering Institute
Publication CERTCC-2014-24
This paper was published in the IEEE 3rd International Congress on Big Data

June 2014

Executive Summary

A large organization can generate over ten billion network flow records per day, a high-velocity data source. Finding useful, security-related anomalies in this volume of data is challenging. Most large network flow tools sample the data to make the problem manageable, but sampling unacceptably reduces the fidelity of analytic conclusions. In this paper we discuss SiLK, a tool suite created to analyze this high-volume data source without sampling. SiLK implementation and architectural design are optimized to manage this Big Data problem. SiLK provides not just network flow capture and analysis, but also includes tools to analyze large sets and dictionaries that frequently relate to network flow data, incorporating higher-variety data sources. These tools integrate disparate data sources with SiLK analysis.

1 Introduction

SiLK (System for Internet-Level Knowledge) [1] is a comprehensive open-source tool suite for storing and analyzing metadata on network traffic traces, called network flow records. SiLK is optimized for security analysis. Today, large organizations routinely generate over 10 billion network flow records daily, which summarize 250-280 TB of network traffic. SiLK permits analysts to make relevant,

©2014 Carnegie Mellon University

Software Engineering Institute | Carnegie Mellon

reliable cyber-security conclusions about such traffic volumes efficiently across years of historical data without sampling the traffic.

This paper presents an overview of the algorithmic and architectural designs that enable analysis of such large data sets. Development work on SiLK began in the early 2000s, and the first open-source version was published on December 22, 2003 by the CERT program of the Software Engineering Institute at Carnegie Mellon University. Thus, this is an overview of over 10 years of development; source code of all aspects discussed are available with documentation from `http://tools.netsa.cert.org/silk`.

The primary goal of the SiLK tools is enhancing the situational awareness of the analyst and scaling to function on the largest instrumented networks, drawing heavily on the UNIX philosophy of smaller tools flexibly interconnected to perform complex tasks. This goal leads to practical requirements such as reduction of worst-case analysis times, so arbitrary analyst queries can compute, for example.

Network routers create flow records by aggregating packets observed within a certain time frame and the same values in the following fields: IP protocol, source and destination IP address, and source and destination port—termed the five-tuple. IP addresses are a key aspect of all network traffic, they are the globally-routable identifier for endpoints. Packets with the same five-tuple represent one communication stream. Flow records contain summary data about the traffic stream, including start time, duration, the five-tuple, TCP flag information, total bytes and packets, some routing information and, optionally, application identifiers.

The paper describes our tools to work with this network flow data; it is organized as follows: Section 3 describes the challenges of data collection and storage at scale and SiLK's solutions to these challenges; Section 4 describes analytic goals, how SiLK is optimized to enable those goals, and presents some timing statistics compared to other common tools; Section 5 describes how SiLK integrates certain heterogeneous data in network flow analysis and presents data demonstrating SiLK's rapid processing of large sets of IP addresses; and Section 6 presents concluding remarks.

2 Related Work

Network traffic analysis tools have a long history. The de facto tool for passive packet capture is `tcpdump` [2], and the de facto human interface for full-packet analysis is the open-source Wireshark [3]. Both of these tools provide full details of the observed network traffic – which can quickly overwhelm storage and analysis resources on large networks. SiLK data summarizes full packet capture, permitting greater scale. Active analysis of networks is possible, via tools such as nmap [4],

however SiLK is a passive network capture and analysis tool suite, and thus attacks different questions than active tools.

There are other open-source passive capture tool suites, such as Argus [5]. Argus stores more data about a network flow than SiLK, and like SiLK provides analysis capability as well as collection, however in order to calculate the additional metadata an Argus sensor must see both incoming and outgoing communications – that is, the routing must be symmetric. Internet routing is not guaranteed to be symmetric; in fact, Internet architecture often encourages asymmetric routing. Unidirectional flows are robust in case of asymmetric routing. Some formats create unidirectional flows, like Cisco NetFlow v5, and some flowmeters, like Argus, create bidirectional flows that describe both sides of the communication in one record; some formats, like NetFlow v9, are configurable for either. One SiLK flow record is strictly unidirectional, a second record is stored for the other side of the communication, which provides robustness for more complex sensor networks.

There are also various proprietary solutions for passive network capture, notably by Cisco, who inspired the standard IPFIX format [6, 7] – but also by most routing hardware vendors. These solutions are embedded in routers, which reduces flexibility: the SiLK suite flowmeter can collect data at more flexible sensor points. These proprietary collection solutions generally do not provide tools for analysis. The SiLK suite encompasses both collection and analysis, and the SiLK analysis tools can ingest data from routers or the native SiLK flowmeter.

3 Data capture and storage

The data capture and storage solutions used in SiLK are designed to optimize analysis. The two primary features to support security analysis are unsampled traffic data and time-centric queries. Within these constraints, SiLK is engineered to be reliable despite the velocity and volume of network data that must be processed and stored.

One central tenet of network security analysis, as opposed to traffic engineering, is that the analyst cannot use a statistical representation of the traffic. Traffic engineers can sample packets at 1:1000 and make viable conclusions. Security analysts do not have this luxury – we miss relevant attacks.[1] Therefore, SiLK emphasizes unsampled network flow; we have achieved unsampled monitoring at 20 Gbps.

Data storage must be able to account for widely variable queries while constantly ingesting new data. We have been able to design a storage solution capable

[1] E.g., 10 years of annual FloCon proceedings support this:
`www.flocon.org/presentations.html`

of providing reliable query times even in the worst case while only requiring the data to be written once. Time is an essential feature of network flow data analysis, and we leverage this analysis goal in the storage solution.

3.1 Capture

There are many flowmeters which convert network traffic to many network flow formats; SiLK then ingests network flow and makes SiLK-format records. In the simplest case, SiLK runs on the same machine as the flowmeter; this is the case we introduce here. There is a more complicated scenario in which multiple sensors capture data in a distributed sensor network which transmit the data to a single SiLK storage instance, for details see [8].

Running SiLK on a single machine uses the following configuration. The rwflowpack daemon collects flow records, converts the foreign flow formats such as NetFlow or IPFIX to the SiLK format (trimming some information to optimize performance), and categorizes the SiLK flow records to determine where on disk they will be stored. Finally, rwflowpack writes the SiLK records into binary flat files where each file represents a specific category, sensor, and hour. This whole process is referred to as *packing* [8]. *Packing logic* refers to the decision process that rwflowpack uses to categorize a flow, described in Algorithm 1 [9].

Since SiLK handles only uni-flows, the system splits traffic into incoming-bins and outgoing-bins. This partition helps match an incoming flow with the related outgoing flow. Flows are further subdivided to assist analysis, storage, and troubleshooting. The primary subdivisions break incoming and outgoing into unrouted and blocked by ACL policy (innull, outnull), and by protocol/port combinations (e.g., inweb, outicmp). For collectors on machines that act as both gateway and internal router it is normal to observe internal to internal (int2int) traffic. External to external (ext2ext) traffic typically indicates a configuration error in SiLK or upstream, and "other" traffic almost always represents a configuration error. The packing logic is completely configurable using plug-ins, and so arbitrary types could be used if desired.

SiLK can leverage our open-source YAF (Yet Another Flowmeter) [10] to produce network flows. YAF takes raw network traffic and converts it to the RFC-standard flow format (IPFIX [6]); with some specialized hardware YAF can process a 20 Gbps network connection without sampling. YAF is not technologically required, SiLK can ingest data from other source types. However, to our knowledge 20 Gbps is as fast as any existing open-source flowmeter can process unsampled flow. Furthermore, the flowmeter is the limiting factor in collection – SiLK keeps up with processing and storing the flows once they are produced.

Algorithm 1 `rwflowpack` packing logic

```
if flow in discarded:
 continue
elif source-network is external || sIP in external-ipblocks || in_interface in
      external-interfaces:
  if destination-network is null || dIP in null-ipblocks || out_interface in null-
      interfaces:
   pack as innull
  elif destination-network is internal || dIP in internal-ipblocks || out_interface
      in internal-interfaces:
   pack as in, inicmp, or inweb by protocol/ports
  elif destination-network is external || dIP in external-ipblocks || out_interface
      in external-interfaces:
   pack as ext2ext
  else:
   pack as other
elif source-network is internal || sIP in internal-ipblocks || in_interface in
      internal-interfaces:
  if destination-network is null || dIP in null-ipblocks || out_interface in null-
      interfaces:
   pack as outnull
  elif destination-network is external || dIP in external-ipblocks || out_interface
      in external-interfaces:
   pack as out, outicmp, or outweb by protocol/ports
  elif destination-network is internal || dIP in internal-ipblocks || out_interface
      in internal-interfaces:
   pack as int2int
  else:
   pack as other
else:
 pack as other
```

3.2 Storage

SiLK data is stored simply but elegantly, providing good performance for both storage and analysis. This section presents the storage decisions made, and then the rationale. First we describe the record format, and then the on-disk file-storage partitioning decisions.

The SiLK binary record format is optimized for network flow. A fully-expanded,

standard SiLK record containing only IPv4 addresses is 52 bytes; with IPv6 each record is 88 bytes. The full format is described in Table 1. To further reduce record size, SiLK typically stores values in as few bits as possible and can remove fields sent by third-party flowmeters which are not often used in security analysis, such as SNMP interface or next-hop IP. Finally, standard compression (zlib, LZO) can further reduce file size.

Table 1: *SiLK default record format (IPv4 and IPv6) [11]*

IPv4 Bytes	**IPv6 Bytes**	**Field**	**Description**
0-7	0-7	sTime	Flow start time as milliseconds since UNIX epoch
8-11	8-11	duration	Duration of flow in milliseconds (max 49 days)
12-13	12-13	sPort	Source port
14-15	14-15	dPort	Destination port
16	16	protocol	IP protocol
17	17	class,type	flowtype
18-19	18-19	sensor	Sensor ID as set by SiLK packer
20	20	flags	Cumulative OR of all TCP flags
21	21	initialFlags	TCP flags in first packet or 0
22	22	sessionFlags	Cumulative OR of all TCP flags sans first packet or 0
23	23	attributes	Various attributes of the record
24-25	24-25	application	Guess as to the content the flow
26-27	26-27	n/a	Unused
28-29	28-29	in	Router incoming SNMP interface
30-31	30-31	out	Router outgoing SNMP interface
32-35	32-35	packets	Total packets in the flow
36-39	36-39	bytes	Total bytes in the flow
40-43	40-55	sIP	Source IP
44-47	56-71	dIP	Destination IP
48-51	72-87	nhIP	Router Next Hop IP

Each file of binary SiLK records has a small header, usually between 24 and 88 bytes. The file header can be longer if it contains metadata about the commands used to create the file. Reading files in this binary format is efficient. For example, as displayed in Figure 1 reading a sample file of 28,107,300 records takes a single machine about 6 seconds for IPv4 addresses or 9.5 seconds if IPv6 is enabled.

Format	Compression	File Size (B)	B/sec	Median t (s)
generic	none	1,461,583,708	102,445,062	14.267
generic	lzo	662,272,502	100,010,948	6.622
ipv6	none	2,473,449,352	83,166,314	29.741
ipv6	lzo	818,932,420	85,252,177	9.606

Figure 1: *Median time to read and process one SiLK file of 28,107,300 records with* `rwfilter`*, considering different file formats on a commodity RHEL5 machine with 4 3.10 GHz cores and 4 GB of RAM. Each test was run seven times, table reports median time to completion as reported by* `time` *command build in to* `zsh` *4.2.6.*

Since the process is I/O bound, although compression uses processing resources it is a net benefit to read time.

On disk, SiLK data is files organized via partition by type, time, and sensor. Within the files, the records are stored in the order they arrived at the final storage location. The storage hierarchy is configurable; the default form is `SILK_DATA_ROOTDIR/TYPE/YYYY/MM/DD/`. `TYPE` is decided by the packing logic in Algorithm 1 [8, p. A.4].

SiLK data is partitioned primarily by time because this organization supports the majority of queries. Short term reviews of recent events are obviously useful. Even for wide-ranging historical queries, the analyst is primarily interested in the time period as the salient factor.

While there are a number of indexes that might be used effectively instead of partitioning by type, time, and sensor, one drawback of any indexing scheme is the overhead required to build and use indexes. Which dimensions of SiLK data a query is based on varies widely, and many of these dimensions are rather well-filled. For example, while TCP port numbers are useful analytically, almost all TCP port numbers will appear in every hour of data in a sizable installation. With a very large installation, even IP addresses have this problem. With such a large installation, narrowing queries by specific internal network segments is a more useful strategy. The hierarchy can be partitioned by sensor to this end while still allowing unusual traffic to be observed, such as routing past unexpected sensors.

Because network flow data are spread fairly evenly within each partition, the utility of indexes is decreased. Even when random access is possible, the cost of accessing the index and randomly accessing the partition is considerable, and a full scan of the partition has competitive performance.

This can be improved if the partitions are clustered on a specific field of interest, but every additional clustering order requires another full copy of the data, limiting the amount of archival data that may be stored. It is common for breaches to go

unnoticed for months, if not years [12, 13], and so limiting archival data below this threshold is not acceptable analytically if the data will maintain its utility.

All of these additions or indexes would increase the amount of overhead data that must be processed and examined, both at the time new data arrives, and at the time data is queried. By focusing instead of decreasing the overhead of the system, SiLK makes it possible to optimize for near-real-time streaming processing of very large amounts of uniform data, which is only ever written once.

One trade-off is that SiLK does not have especially fast access patterns for any specific network flow query; however worst case performance is minimized. Therefore, any query will run in predictable time, based on the duration of the network traffic trace to be examined. Since queries tend to vary widely, it is especially important to minimize worst-case query times. Furthermore, a rough index using IP set files (see Section 5) for the source and destination IP addresses observed in each hour's worth of files can be used to limit full scans of files. Figure 3 demonstrates that `rwset` can extract such IP sets quickly.

In summary, full scans of the data are generally considered resource intensive compared to random access—but this is only true if queries are either highly selective or highly predictable. In the environment of computer network security analysis, for which SiLK is optimized, queries rarely have these properties. As a result, the strategy of optimizing for full scans of the data partitioned by time range has proved in our experience to be most usable for analysts and provided excellent results. In contrast, traditional database engines usually focus on performance and reliability when dealing with data that changes over time, and must be able to answer predictable, highly-selective queries very quickly. As elaborated above, these are not features of network security analysis of flow data.

4 Analyzing data

The CERT program has been analyzing network flow data for at least as long as we have been writing SiLK. There is strong feedback between the tools' capabilities and the analysts' needs. The handbook for SiLK analysis is large [14] – this section only can highlight a few tools. For examples of operational analysis with SiLK, see [15, 16]. This synergy between development and analysis has created an effective, refined set of tools. There are dozens of specialized command-line tools for flow analysis in the SiLK tool suite; we will focus on the central tool, `rwfilter`.

Most simply, `rwfilter` serves two purposes: an interface to the SiLK data store for retrieving records, and to partition those records based on flexible input criteria into "pass" and "fail" data streams. The high-level data flow through one `rwfilter` call is described in Algorithm 2. The algorithms in `rwfilter` are not

Algorithm 2 Basic `rwfilter` logic

```
select files to read([sensor],[date],[hour],[type],[file name])
read user partitioning criteria
read user output options([output_pass],[output_fail])
for file in files:
 for records in file:
  if record matches criteria:
   output_pass.write(record)
  else:
   output_fail.write(record)
```

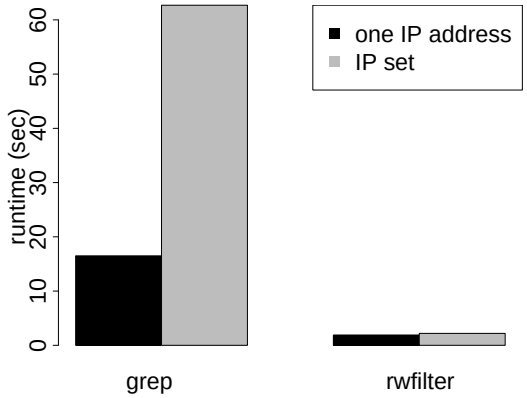

Figure 2: *Timing results for searching through 12,391,000 flow records for either one IP address or a set of 43,096 random IP addresses, using `rwfilter` and `grep` on the same machine (64 2 GHz processors, 192 GB RAM).*

complex in themselves, but managing the efficient and robust implementation of the 112 switches necessary to support even relatively simple analysis of network flow at scale is not trivial.

The SiLK tools are much more efficient than the standard Linux text-based software tools for analyzing network flow. Figure 2 displays the timing results for searching through 12+ million flow records for both a single IP address and a set of IP addresses. The SiLK tools not only are faster, they are much more consistent: both queries take about two seconds. In the case of looking for just one IP, the Linux tools take about 8 times longer to run. But as the query becomes more complex, the optimizations for SiLK are more obvious, as the Linux tools take 28 times longer to run.

In addition to improved performance, `rwfilter` also facilitates big analysis in other ways: query metadata and iteration. `rwfilter` stores the commands used to create the output in the header of the output file. These breadcrumbs are invaluable for analysts to track their thought process. Network analysis is complex, and often requires iteration. `rwfilter` results are frequently processed by further `rwfilter` queries to refine results. The iterative characteristics of `rwfilter` are based on the fact it both ingests and outputs SiLK binary files, and this feature permits the analyst to iteratively grow complex queries efficiently. The breadcrumbs in the query metadata ensure the analyst remembers where they are in the iteration process.

The tool suite includes several other useful tools, such as `rwstats`, `rwcut`, `rwsort`, and `rwuniq`. `rwstats` provides flexible summary statistics based on any combination of fields in a SiLK record, helping answer such analysis questions as, for example, which host IP address sent the most DNS traffic yesterday. `rwcut` produces human-readable output. `rwsort` sorts SiLK records based on user-supplied fields. `rwcut`, `rwsort`, and `rwuniq` are analogous to the standard UNIX utilities `cut`, `sort`, and `uniq` used for text processing. These SiLK tools support flexible, robust analysis of big network data while maintaining the practical focus by keeping the data binary and compact as much as possible so the analysis is computationally practical.

The SiLK tool suite has a variety of tools for manipulating IP addresses, both in respect to network flows and independently. These SiLK tools are often orders of magnitude faster than text-processing tools. Figure 3 displays the completion time for two different tasks. The left side is a command to extract all the unique source IP addresses from 12,391,000 flow records; the same file and machine as for Figure 2. Note the log scale – the SiLK tool rwset completes in 2.8 seconds, while the other tools require 818 seconds (13+ minutes). On the right of Figure 3 SiLK shows an even bigger advantage over the common tool `tcpdump` – 433 seconds for SiLK versus 71242 (19+ hours) to extract and count the number of packets in a 39 GB file.

5 Additional Data Structures

Network flow analysis is the core competency of the SiLK tool suite, but SiLK's largest advantage may be the processing and integration of different data types related to network flow. Since the SiLK capture and storage solutions do no sampling, the tools to work with the related data must also be improved in order to cope with the large data volumes. Primarily, these tools focus on IP address processing and labeling such as sets and prefix maps. The IP set libraries are also available in a stand-alone version that does not require the rest of the SiLK tool suite.

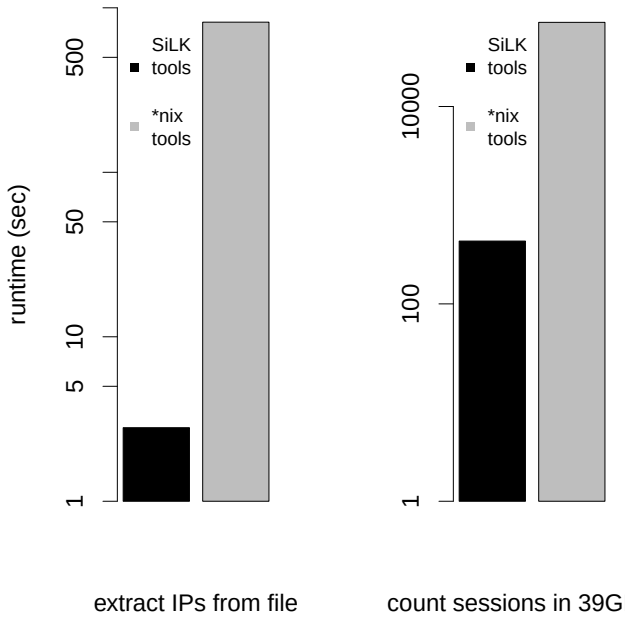

Figure 3: *Two cases where SiLK tools are orders of magnitude more efficient than non-specialized tools. Left side is time required to extract all the unique source IP addresses from* 12,391,000 *flow records, using* `rwset` *or the *nix tools* `cut | sort | uniq`*. Right side is* `rwp2yaf2silk` *versus* `tcpdump | wc` *in counting packets/sessions in a large full packet capture file. Note the logarithmic scales.*

SiLK IP sets are optimized in order to scale to Internet-sized sets of IP addresses while remaining practical both for long-term storage and when interacting with other large sets or vast amounts of network flow. To this end, IP sets use an in-memory data structure optimized for access speed and a different, on-disk representation optimized for minimum size. The on-disk representation is particularly good for sparse sets; as of SiLK 3.7.0, version 4 IP set files can be created with a slightly different format that is better suited for densely populated sets while retaining most of the advantages for sparse sets in the version 2 IP set format.

In RAM, IP sets are initialized as an array of 65,536 (2^{16}) pointers, one for each possible /16 CIDR block[2] in IPv4 space. When the first address in a /16 is loaded into memory, a bitmap of 65,536 bits is allocated to the pointer, where each bit represents one of the addresses in the /16. The bitmap is represented as an array

[2]Classless Inter-Domain Routing (CIDR) prefixes note network size [17]. For a /n network, IPv4 size is 2^{32-n} and IPv6 size is 2^{128-n}.

Algorithm 3 Reading and writing IP sets

```
read(file.set):
 for each cidr_24_block in file.set:
  if cidr_16_block of cidr_24_block not seen:
   allocate cidr_16_block in mem.set
  for each 32_bit_word in cidr_24_block:
   cidr_16_block[cidr_27_block] := 32_bit_word
write(file.set):
 write(file_header)
 for each cidr_16_block in mem.set:
  data := read(8 cidr_27_blocks)
  if data:
   write(cidr_24_block)
```

of 2,048 32-bit unsigned integers, where each 32-bit integer is a bitmap of a /27 CIDR block.

However, on disk each populated /24 CIDR block (made up of 256 addresses) is represented by a block of 9 32-bit words. The first word encodes the base IP address (i.e., a.b.c.0 where $0 \leq a,b,c \leq 255$) and the other 8 words form a 256-bit array to encode the 256 addresses in the /24; an IP address is in the set if its bit is set. The functions for reading and writing IP sets handle the conversion between these two formats, as described in Algorithm 3.

The disk format achieves size savings; the memory format improves access speed. A sparse IP set size is reduced greatly by the on-disk format. There are many 0's in a sparse set; the disk format does not store these zeros, but is structured in such a way that the information is retained.

On the other hand, while the initial (Version 2) IPset disk format is good for sparse arrays and sets, it is bad for dense arrays, since a full /8 would have 65,536 /24s of 288 bits each. This can lead to unnecessarily large set files. Version 4 IPset files (introduced in SiLK 3.7.0) attempt to fix the problems of storing dense IPsets with the following rules:

- For CIDR blocks that contain 256 or more IPs, the new format contains the IP and a single byte for the netblock (CIDR) prefix. Space savings are large: "10.0.0.0/8" is reduced from more than 2×10^6 bytes to five.

- For smaller CIDR blocks, if noncontiguous IPs or CIDR blocks appear, the file contains the base IP address, a special value for the prefix, and a 256-bit bitmap (similar to that in SiLK-2) indicating which IPs are active. Thus, to store 10.11.12.13 and 10.11.12.15, the file contains the equivalent

of "10.11.12.0 129". The 129 indicates that a bitmap follows. Bits 13 and 15 are set in that bitmap.

- If a single IP or CIDR block appears in the IPv4/24, the file may use either representation.

Algorithm 4 Basic set operations with SiLK IP sets

```
union(sets):
  for each cidr_16_block in set1:
    if cidr_16_block in set2:
      for each bitmap in cidr_16_block:
        set2[bitmap] |= set1[bitmap]
    else:
      memcpy(set2[bitmap], set1[bitmap])
  output set2
intersect(sets):
  for each cidr_16_block in set2:
    if cidr_16_block not in set1:
      free set2[cidr_16_block]
    else:
      has_content = 0
      for each bitmap in cidr_16_block:
        set2[bitmap] &= set1[bitmap]
        has_content |= set2[bitmap]
      if has_content == 0:
        free set2[cidr_16_block]
  output set2
difference(sets):
#compute set2 = set2 - set1
  for each cidr_16_block in set2:
    if cidr_16_block in set1:
      has_content = 0
      for each bitmap in cidr_16_block:
        set2[bitmap] &= ~(set1[bitmap])
        has_content |= set2[bitmap]
      if has_content == 0:
        free set2[cidr_16_block]
  output set2
```

Set math with these optimized data structures is much faster. The functions for intersection, union, and set difference are described in Algorithm 4. The performance for these operations on several sizes and heterogeneity of sets is displayed in Figure 4 with detailed results displayed in Table 3. Each reports the mean and median of seven runs of each operation. These tests use certain sets to test SiLK tool performance which are defined in Table 2.

Table 2: *Details of sets used in Figure 4 and Table 3*

Name	Description	# of IPs	file size (B)
odd	$x.x.x.y \in 0.0.0.0/0 : y\%2 = 1$	2,147,483,648	603,979,833
even	$x.x.x.y \in 0.0.0.0/0 : y\%2 = 0$	2,147,483,648	603,979,834
all	$x.x.x.y \in 0.0.0.0/0$	4,294,967,296	603,979,833
rand	randomly selected	43,096	1,549,393
10K	randomly selected	10,000	variable
1M	randomly selected	1,000,000	variable
RFC1918	unrouted IPs [18]	336,723,712	938

Prefix maps are a data format used by SiLK to map bit strings (usually IP addresses) to an associated text value. The base of the format is the "trie" data structure, "a tree for storing strings in which there is one node for every common prefix" [19]. Bit strings are used because of their compact representation of the data.

SiLK uses tries because they are efficient at storing data in which many keys with the same prefix represent the same value. This is common for IP addresses where large CIDR blocks, perhaps 2^{24} IP addresses, may map to a single value. A trie compactly represents this mapping as a single entry.

Algorithm 5 describes the method for creating a prefix map. The trie nodes are stored as entries in a contiguous array of 32-bit integers, which reduces memory allocation overhead. Every node is two 32-bit numbers, one for each the left and right child of the node. If the most significant bit of this integer is 0, this value is the array index of the child node in the trie. If the most significant bit of the number is 1, the child is a leaf and the lower 31 bits are the integer value at the leaf. For simplicity, the implementation assumes that the input keys are sorted from most general to most specific.

When searching a prefix map for a key (e.g. IP address), the search starts from the root node and the first bit of the key. The search follows the left branch if the next bit is a zero and the right branch if it is a one. When the search finds a leaf, the value of that leaf is returned—the leaf represents the value shared by all bit strings which begin with the bits so far. Algorithm 6 describes the search process. Finally the search may look up the integer value in the dictionary to produce the

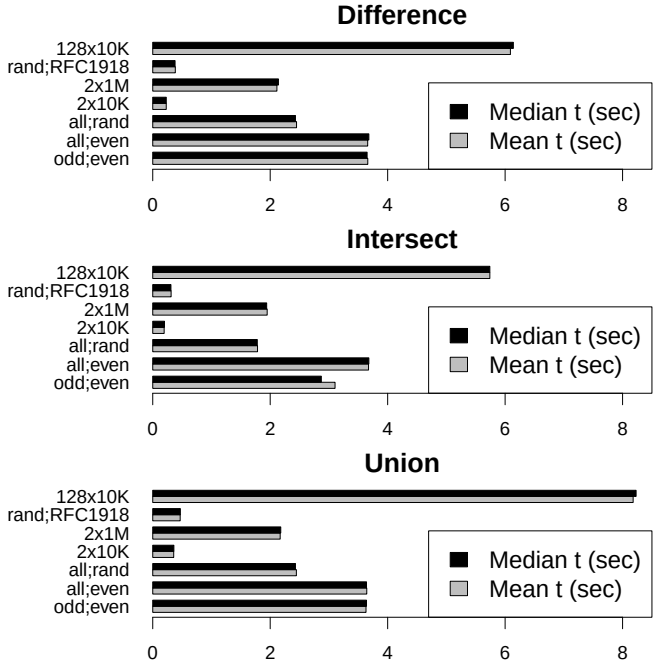

Figure 4: *Compute time (sec) of various set operations. Tests were run on a machine with 24 2.67 GHz processor cores and 64GB RAM. Seven runs per test. 128x10K means 128 sets each of* 10^4 *IP address; likewise 2x1M is two sets of* 10^6 *IP address each.*

text value for that prefix. Finding the value for an n-bit key takes at most n steps; this constrained maximum is in keeping with the design principle of predictable and reasonable worst-case processing time.

This binary trie structure enables Internet-scale labeling of important analysis features, such as Autonomous System Number (ASN), which indicates control of the IP address announced via BGP [20]. The CERT program makes prefix maps of this IP-ASN relationship freely available [21].

6 Conclusions

We have presented some important features of SiLK, a robust and efficient open-source tool suite for analyzing big data related to network traffic analysis practically and without sampling. We know of no other tool suite with all these features. The tools are under active development; substantive improvements to SiLK, such

Table 3: *Compute time (s) and memory usage (kB) of set operations*

Sets	Ops	Mean *t*	Median *t*	Mean RAM	Median RAM
odd, even	∩	3.104	2.87	1,049,737	1,049,737
odd, even	∪	3.631	3.64	1,049,801	1,049,801
odd, even	−	3.661	3.65	1,049,801	1,049,801
all, even	∩	3.674	3.68	1,049,801	1,049,801
all, even	∪	3.644	3.64	1,049,801	1,049,801
all, even	−	3.660	3.68	1,049,801	1,049,801
all, rand	∩	1.784	1.78	778,249	778,249
all, rand	∪	2.446	2.43	778,249	778,249
all, rand	−	2.449	2.43	778,249	778,249
2 x 10K	∩	0.194	0.20	149,465	149,465
2 x 10K	∪	0.357	0.36	213,089	213,089
2 x 10K	−	0.230	0.23	149,529	149,529
2 x 1M	∩	1.949	1.94	1,049,801	1,049,801
2 x 1M	∪	2.170	2.18	1,049,801	1,049,801
2 x 1M	−	2.114	2.14	1,049,801	1,049,801
128 x 10K	∩	5.740	5.74	9,569,576	9,569,576
128 x 10K	∪	8.180	8.23	10,019,576	10,019,576
128 x 10K	−	6.090	6.14	9,569,640	9,569,640
rand, RFC1918	∩	0.313	0.31	295,073	295,073
rand, RFC1918	∪	0.467	0.47	316,441	316,441
rand, RFC1918	−	0.384	0.38	295,073	295,073

as for SiLK 4, would include support more flexible data types, back-end storage flexibility, and improving parallelization.

From collection to storage to analysis, SiLK presents a coherent framework for practical analysis on modern large networks. Researchers investigating phenomena unrelated to network traffic traces may find use in the efficiency of set math and prefix map capabilities for 32-bit and 128-bit integers (IPv4 and IPv6 addresses). Those working with full-packet-capture network data will find network flow a useful index into the data. Further details, including source code, are available from `http://tools.netsa.cert.org/silk`.

Acknowledgment

This material is based upon work funded and supported by the Department of Defense

Algorithm 5 Creating and building a SiLK Prefix Map

```
insert(IP/cidr, value):
 if (root == null):
  root = new leaf()
  root->value = default
 node = root
 n = 0
 while (n < cidr):
  ++n
  if (node->is_leaf == true):
   # make it an interior node
   node->left = new leaf()
   node->left->value = node->value
   node->right = new leaf()
   node->right->value = node->value
   node->is_leaf = false
  if (nth bit of IP == 0):
   node = node–>left
  else:
   node = node–>right
 if (node->is_leaf == false):
  node->left = null
  node->right = null
  node->is_leaf = true
 node->value = value
```

Algorithm 6 Searching a SiLK Prefix Map

```
search(IP):
 node = root
 n = 0
 while (node–>is_leaf == false):
  ++n
  if (nth bit of IP == 0):
   node = node–>left
  else:
   node = node–>right
 return node–>value
```

under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation

of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This material has been approved for public release and unlimited distribution.

Carnegie Mellon®, CERT® and CERT Coordination Center® are registered marks of Carnegie Mellon University.

DM-0001043

References

[1] CERT/NetSA at Carnegie Mellon University, "SiLK (System for Internet-Level Knowledge)," [Accessed: Feb 4, 2014]. [Online]. Available: http://tools.netsa.cert.org/silk

[2] Tcpdump/Libpcap, "TCPDUMP & Libpcap homepage," [Accessed: Mar 24, 2014]. [Online]. Available: http://www.tcpdump.org/

[3] Wireshark Foundation, "Wireshark homepage," [Accessed: Mar 24, 2014]. [Online]. Available: http://www.wireshark.org/

[4] G. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide To Network Discovery And Security Scanning*. Nmap Project, 2011.

[5] QoSient LLC, "Argus: Auditing Network Activity," [Accessed: Mar 24, 2014]. [Online]. Available: http://qosient.com/argus/index.shtml

[6] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101 (Proposed Standard), Tech. Rep. RFC 5101, Jan. 2008.

[7] B. Claise, P. Aitken, and N. Ben-Dvora, "Cisco Systems Export of Application Information in IP Flow Information Export (IPFIX)," RFC 6759 (Informational), Tech. Rep. RFC 6759, Nov. 2012.

[8] CERT/NetSA at Carnegie Mellon University, "SiLK Installation Handbook; SiLK-3.8.1," Jan 30 2014, [Accessed: Feb 9, 2014]. [Online]. Available: http://tools.netsa.cert.org/silk/install-handbook.pdf

[9] ——, "SiLK: rwflowpack man page – packing logic," [Accessed: Feb 9, 2014]. [Online]. Available: http://tools.netsa.cert.org/silk/packlogic-twoway.html#Packing-logic-code

[10] C. M. Inacio and B. Trammel, "Yaf: Yet another flowmeter," in *Large Installation Systems Administration (LISA)*. San Jose, CA: USENIX, 2010. [Online]. Available: https://www.usenix.org/legacy/events/lisa10/tech/slides/inacio.pdf

[11] CERT/NetSA at Carnegie Mellon University, "SiLK FAQ: SiLK Flow file format," [Accessed: Feb 9, 2014]. [Online]. Available: https://tools.netsa.cert.org/silk/faq.html#file-formats

[12] "2012 data breach investigations report (DBIR)," Verizon, Tech. Rep., 2012. [Online]. Available: http://www.verizonenterprise.com/DBIR/2012/

[13] "2013 data breach investigations report (DBIR)," Verizon, Tech. Rep., 2013. [Online]. Available: http://www.verizonenterprise.com/DBIR/2013/

[14] T. Shimeall, S. Faber, M. DeShon, and A. Kompanek, "Analysts' handbook: Using SiLK for network traffic analysis," Software Engineering Institute, CERT Program, Pittsburgh PA, Tech. Rep., 2010. [Online]. Available: http://tools.netsa.cert.org/silk/analysis-handbook.pdf

[15] CERT/NetSA at Carnegie Mellon University, "Proceedings – FloCon," [Accessed: Feb 21, 2014]. [Online]. Available: http://www.cert.org./flocon/proceedings.html

[16] A. Whisnant and S. Faber, "Network profiling using flow," Software Engineering Institute, CERT Program, Pittsburgh PA, Tech. Rep. CMU/SEI-2012-TR-012, 2012. [Online]. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1686&context=sei

[17] Y. Rekhter and T. Li, "An Architecture for IP Address Allocation with CIDR," RFC 1518 (Historic), Tech. Rep. RFC 1518, Sep. 1993.

[18] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918 (Best Current Practice), Tech. Rep. RFC 1918, Feb. 1996.

[19] P. E. Black, "Dictionary of algorithms and data structures," V. Pieterrse and P. E. Black, Eds., Feb 22 2011, [Accessed: Feb 13, 2014]. [Online]. Available: http://xlinux.nist.gov/dads/HTML/trie.html

[20] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), Tech. Rep. RFC 4271, Jan. 2006, updated by RFCs 6286, 6608.

[21] CERT/NetSA at Carnegie Mellon University, "CERT/CC Route Views Project Page," [Accessed: Feb 13, 2014]. [Online]. Available: http://routeviews-mirror.cert.org.