This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

A TCP Connection Establishment Filter: Symmetric Connection Detection

Brad Whitehead and Chung-Horng Lung
Systems and Computer Engineering
Carleton University, Ottawa, Canada
{bwhitehe, chlung}@sce.carleton.ca

Peter Rabinovitch
Research and Innovation
Alcatel-Lucent, Ottawa, Canada
peter.rabinovitch@alcatel-lucent.com

***Abstract*— Network measurement at 10+Gbps speeds imposes many restrictions on the resource consumption of the measurement application, making any filtering of input data highly desirable. Symmetric Connection Detection (SCD) is a method of filtering TCP sessions, passing only those sessions which become fully established. SCD can benefit network monitoring applications that are only interested fully established TCP connections by reducing processing requirements. Incomplete connection attempts, such as port scanning attempts, simply waste resources in many applications if they are not filtered. SCD filters out unsuccessful connection attempts using a combination of Bloom filters to track the state of connection establishment for every flow passing through a network device. Unsuccessful flows can be filtered out to a very high degree of accuracy, depending on the size of the Bloom filter and traffic rate, 99.5% is typical. Resource consumption, both memory and CPU is low. The core SCD algorithm is designed to work in high-speed routers, in real-time, and at line speed. Using an upper bound of 32k bytes of RAM our experimental results indicate 99+% accuracy with 900,000 active flows.**

I. Introduction

Network traffic analysis applications intended for network engineering or other applications by ISPs may require computationally intensive processing of flow records. Processing the flow records generated by a typical OC-192 (10Gbps) edge router typically requires either large computational resources, or techniques which reduce the computational load at the expense of accuracy. In this paper we advocate a third approach to this problem; focus available computing resources on flows which impact directly on the desired measurement by pre-filtering incomplete flows.

Existing solutions for monitoring network traffic have the common problem of scalability. The Real-Time Flow Monitoring (RTFM) specification describes a fine-grained, flexible, and programmable architecture for monitoring network traffic [5]. This architecture processes every packet, matching it to an existing flow record or creating a new record if the flow is not found. Many different statistics can be calculated based on the resulting records, with no loss of accuracy. However, architectures based on per-flow association of packets, such as RTFM, can not scale to high-speed links such as OC-192 or OC-768 [10]. Currently specialized hardware is required to support lower speed links such as OC-48 [6].

Ensuring scalability to 10+Gbps speeds requires a careful approach to the design of the monitoring application. The industry standard flow reporting solution, NetFlow, is implemented on edge and core routers using sampling of packets to reduce processing and memory requirements [16]. The simplest form of sampling selects only one out of every n packets, thus introducing substantial inaccuracy in many network statistics [9]. Several techniques have been proposed to increase the accuracy of sampled flow statistics [8], or improve the accuracy for specific applications [7]. These techniques are limited to solving a specific problem, and can only place an upper bound on the sampling induced inaccuracy.

Scalability is a crucial element of network measurement applications, and yet has proved one of the most strenuous problems to solve. The difficulty of the scalability problem is best demonstrated by the wealth of work presenting complex solutions for otherwise simple problems. These solutions are all intended to provide network measurement at 10+Gbps speeds. Kumar, et al. propose a hardware-based solution which is capable of estimating the number of packets sent on a per-flow basis [13]. They introduce a new type of Bloom filter called a space-code Bloom filter which allows counting and is ideal for a hardware implementation.

Ideally we wish to be able to reduce the processing requirements of traffic analysis without sacrificing accuracy. To this end we describe a filtering technique which is capable of reducing the number of flows, and therefore the computational requirements, by up to 95% for average Internet traffic. Like many other proposed solutions to high-speed network monitoring our solution makes use of a time and space efficient data structure known as a Bloom filter [4]. A Bloom filter is a bit array which supports set membership tests by using k independent hash functions to address k bits in the bitmap. To insert an element the k bits the element hashes to are set to 1, and therefore all k bits will be 1 if the element is a member of the set. One main drawback to Bloom filters that they have a small probability of generating a false positive, which will be further discussed in section III.

Symmetric Connection Detection (SCD) is method of filtering network traffic such that only fully established TCP flows will pass through the filter. SCD uses Bloom filters to maintain minimal state about every TCP connection attempt. The operation of SCD can be summarized as follows; TCP SYN packets are associated to flow identifiers, in a highly compressed format, using two Bloom filters. Once a TCP SYN has been "seen" from both sides of a connection SCD

1-4244-0353-7/07/$25.00 ©2007 IEEE

will report that the connection was successfully established. The unique feature of SCD is its ability to provide very high accuracy while using very small amounts of RAM and CPU time. In section IV we show that using only 32k bytes of memory SCD can achieve 99% accuracy even in adverse conditions (900,00 active flows).

Network monitoring applications such as tracking the duration of TCP flows can be optimized using the pre-filtering provided by SCD to filter out flows which are never fully established. In typical Internet traffic TCP accounts for 95% of traffic, of which 5-10% is SYN packets [19] (also shown in section IV). Reduction in flows can benefit many applications, for example, an application which is tracking the duration of flows only requires access to those flow which are fully established, processing any other flows or SYN packets is a waste of computing and memory resources. In this paper we show that many of the SYN packets seen on the general Internet are not valid connection attempts, but instead are part of DDoS attacks or port scanning. These SYN packets will almost never become established flows. We show in section IV that filtering these flows can reduce the processing requirements of our hypothetical duration tracking software by 95%.

II. Related Work

After an extensive survey, to the best of our knowledge there is no work directly related to filtering incomplete TCP flows out of network flow data in real-time. This is perhaps due to the relative simplicity of the problem when infinite resources are available to filter traffic. In this section, we discuss other work that has laid the ground work for our extension of Bloom filters and approach.

Stateful packet filters are able to track the connection state of TCP sessions, examples of these are [17][5][1]. From the perspective of resource consumption, these stateful filters are equivalent to tracking all flows individually. Storing per-flow state makes these applications very flexible in their feature set, but also requires memory on a per-flow basis. As a result, these applications are unable to process packets at the line speed of a 10Gbps edge router due to, the requirement to use DRAM to store the flow information, and the computational resources required for flow lookup. It will be shown in section IV that even when a stateful filter uses optimization techniques such as a very large hash table to increase flow lookup speed, SCD provides an order of magnitude better performance.

Since SCD focuses on connections that are established, and the opposite problem is detecting connections that are never established, the research into detecting port scanning contains some work that is similar in concept to SCD. However, it should be understood that detecting incomplete connections and reporting complete connections are two different problems. SCD is able to report fully-established connections, but without further processing SCD is not able to report half-open connections. Paxon describes a system called Bro which detects port scans by tracking the number of connection failures for specific hosts [14]. TCP SYN, FIN, and RST packets are used by Bro to track the state of every connection on a per-flow basis (see section III for a brief description of TCP). Tracking per-flow state requires the use of DRAM to store the large amount of state, so Bro is limited to lower-speed networks.

Weaver, Staniford, and Paxon present a method of containing scanning Internet worms by detecting their port scanning attempts [21]. Again this paper focuses on port scan detection, not established connections. The authors mention using Bloom filters as an approximation cache, but not in the context of tracking connection attempts. Their implementation uses an associative cache to track external connections, and requires a notion of internal and external IP addresses, which would result in inefficient operation on edge or core routers.

The use of Bloom filters as a time and space efficient data structure to keep state on set membership is not unique to this paper [3]. A survey of the network applications of Bloom filters is [4]. Attig and Lockwood have shown that a Bloom filter can be implemented in hardware and can scale to OC-192 (10Gbps) speeds [2], a low power strategy is [12]. Attig and Lockwood use a Bloom filter based method to detect patterns in network traffic and report on suspicious flows.

III. Symmetric Connection Detection

Symmetric Connection Detection (SCD) provides a 95% reduction in the number of flows which must be tracked and processed by a per-flow network monitoring algorithm. This reduction is accomplished by reporting when a TCP or other connection-oriented connection attempt is very likely to result in a fully established connection. When used as a filter, SCD is able to filter flows which are never fully established, and therefore pass only those flows which are fully established to a secondary processing algorithm. In this section we give a high-level overview of the operation of basic SCD, and in section III-A we describe an extension to SCD to improve accuracy.

Two fundamental requirements can be identified for any algorithm that implements a filter which passes only complete connections. First, every network packet must be processed and some amount of state must be stored for every connection establishment attempt. Storage of the connection state is necessary for the algorithm to track the connection progress of the endpoints, and determine when a connection is either fully established or is very likely to become fully established. Second, a detection mechanism must decide when the flow establishment process is complete by monitoring or comparing the state of all flows.

Based on these two fundamental requirements the basic operation of SCD is straight-forward to describe. SCD stores the state of all connection attempts and performs a comparison on the connection state to determine when a connection has been established. SCD can report the current connection status in real-time, every time the state of a flow changes. The connection status is reported as a boolean value; true if the flow is now established, and false if it is not yet established. Connection information can then be used to filter or pass packets for that flow to a higher level monitoring system, or the statistics can simply be logged and provided to network operators.

The process of tracking the connection state may be specific to the underlying protocol being tracked. In this section, we assume that the underlying protocol is TCP, and therefore begin with a short description of the TCP connection process and how it relates to SCD. To establish a TCP connection a three-way handshake process takes place; each computer sends a SYN, and the initiating computer sends an ACK to complete the connection [15]. Once the ACK is received the connection process is completes and the TCP session is fully established. Tracking the establishment of a TCP connection therefore requires keeping track of all three states, however this can be simplified to two states with the following observation. From a point in the middle of the route between the computers the receipt of SYN packets from both sides of a connection implies that both computers can reach each other and want to establish a connection, strongly indicating that the connection will be established with a completing ACK. SCD makes use of this observation and defines an established connection as one where both sides have received a SYN from the other side but not necessarily an ACK. Therefore SCD processes only TCP SYN packets, or an average of about 1 in 20 packets (TCP SYN is about 5% of network traffic as discussed in section I).

The problem of tracking connection establishment can now be defined as the following question; when a TCP SYN is received from one side of a connection has the other side already sent a SYN? If so then the connection is established, if not then store the fact that this side of the connection has sent a SYN. To answer this question SCD keeps state on all SYN packets that have been sent and the direction that they were sent in. Direction is determined by comparing the source and destination IP addresses, e.g. if source IP is greater than destination IP then the packet is assigned direction 1, and if source IP is less than destination IP the packet is assigned direction 2. Storing the flows which have sent a SYN in a specific direction could be accomplished through the use of many different data structures, but many potential data structures would lack sufficient performance to be able to keep up with the requirement to perform a search and possibly an insert on every SYN packet. Therefore, the data structure must be time and space efficient, and ideally would support searching and inserts that scale in constant time with the number of items stored. Bloom filters are such a data structure.

SCD is designed to operate in a resource-limited environment, and undergo gradual degradation of accuracy as resources become more limited. This operation is accomplished through the use of Bloom filters. The only data storage required by SCD is the SYN-direction information for each flow. To meet this storage requirement we employ two Bloom filters, one filter for each SYN direction. Bloom filters represent a set that can be tested for membership. Mapping this concept to our problem can be done as follows; when a SYN is received, test the Bloom filter for the opposite direction to see if a SYN was sent from the other side; if so, the connection is established. If a SYN has not yet been received from the other side, then the connection is not yet established, and this is either the first SYN packet in the connection or the other side is not responding. If the flow was not already stored in the filter for its direction it is added.

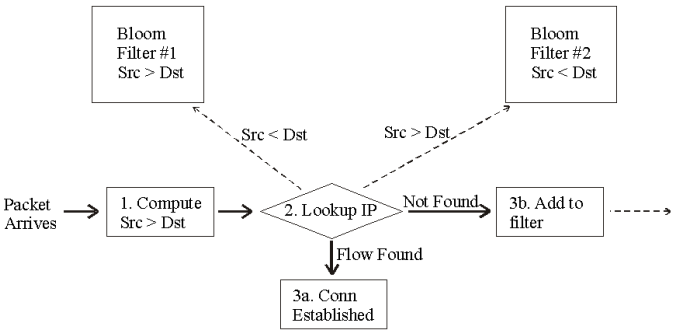

Fig. 1. Flow-Chart of SCD Operation

The use of Bloom filters in SCD introduces the first two parameters of the algorithm. First, the length m of the array of bits comprising the filter can be varied, resulting in a memory usage of 2 * (m/8) bytes for SCD. Second, the number of hash functions, k, used to insert a new entry into the Bloom filter can be varied. The optimal value of k has been calculated to be k = ln(2) * (m/n), where n is the maximum expected number of entries in the filter [4]. The value of k can also affect the computational requirements of SCD since k independent hash functions must be evaluated for every packet. Bloom filters can generate false positives due to the fact that two different flow identifiers may hash to the same k values. The probability of a false positive is given by $f = (1 - e^{-kn/m})^k$.

Figure 1 is a flow chart describing the processing of a TCP SYN packet by SCD, with the following steps:

- **1. Compute Src $>$ Dst:** The source and destination IP addresses are compared as unsigned integers to determine which address is greater, source or destination. If they are equal the packet is assumed to be corrupt, and is ignored.
- **2. Lookup IP:** The Bloom filters are queried to see if a SYN packet was sent in the opposite direction for this flow, e.g., if the incoming packets source IP is greater than the destination IP, then the Bloom filter for the opposite direction (Dst $>$ Src) is queried. The packet is used to generate a number of hashes which are used to query the Bloom filters. The number of hashes used is a parameter.
- **3a. Connection Established:** If the Bloom filter returns a positive result to the query, then it can be concluded that the connection is established, to a high degree of accuracy.
- **3b. Add to filter:** If the Bloom filter returns a negative result the corresponding Bloom filter is updated with the flow, e.g., if the incoming packets source IP is greater than the destination IP, then the flow is added to the Src $>$ Dst Bloom filter.

A potential problem arises when continual operation of SCD is considered. If left unchecked, the Bloom filters representing each direction would eventually become full and false positive error rates would climb to unacceptable levels. To avoid this situation the Bloom filters can be cleared of all their data on a

periodic basis. The length of this period is the third parameter of SCD, the maximum connection time. The maximum connection time describes the maximum time that a TCP connection can take before it is no longer tracked by SCD. If the connection establishment exceeds this time the connection becomes a false negative due to the filters being cleared. A false negative occurs if the connection state is lost when the filters are cleared, because the recorded state verifying that the original SYN was sent is erased, resulting in SCD reporting that the flow was never established (a false negative). This raises a potential issue; the minimum connection time that will report a false negative is potentially 0 if the original SYN packet was received by SCD just before the filters were cleared. We call the minimum time that a TCP connection can take to complete before being lost when the filters are cleared the lower bound of the maximum connection time. This leads us to propose an improvement over basic SCD, dual-filter SCD.

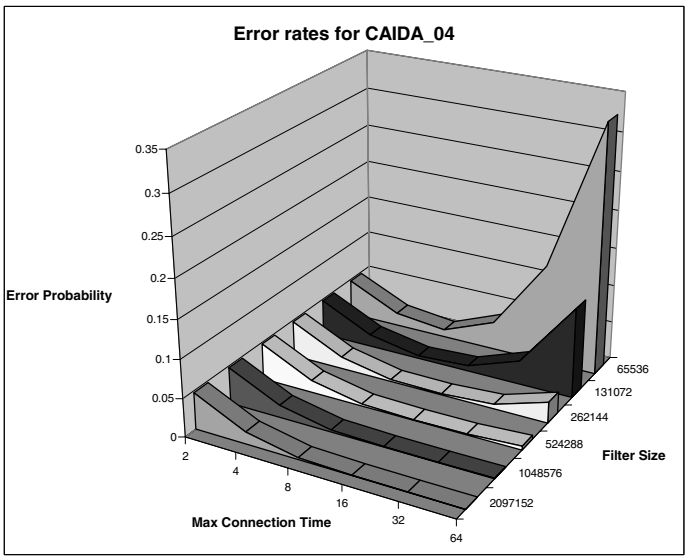

Fig. 2. SCD Performance for Trace C_04

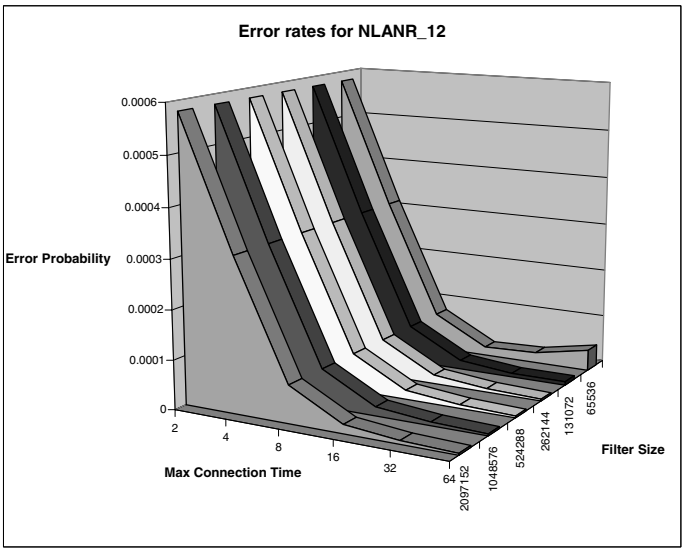

Fig. 3. SCD Performance for Trace N_12

A. Dual-Filter SCD

The connection establishment phase of TCP can range from a few milliseconds to several minutes. This extreme variability in the connection establishment time for TCP is one of the major sources of error in SCD. Connections which take much longer than normal to complete (more than a few seconds) can become false negatives if they exceed the lower bound of the maximum connection time. Dual-Filter SCD reduces the number of errors caused by this variability by raising the lower bound of the maximum connection time from zero to half of the upper bound of the maximum connection time.

Dual-filter SCD modifies basic SCD from one Bloom filter per direction to two Bloom filters per direction. Each Bloom filter contains the state of connection attempts for a non-overlapping portion of the total range of time. For example, if the SCD maximum time is 10 seconds one filter would initially cover the 0-5sec range and the other would track 5-10sec. Flows are moved between filters by an aging process, which will be described below. After 10 seconds of running time the newer filter will contain flows for the past 0-5 seconds, and the older filter would contain flows from 5-10 seconds. During operation of SCD new SYN packets are recorded in the newer filter, and the older filter simply maintains the state of older connection attempts. Upon receipt of a new SYN packet, queries for membership to check if the connection is now established are performed against both the older and newer filters.

As with standard SCD, Dual-Filter requires an aging process to prevent the build up of out of date flow data and maintain accuracy of the filter. Each filer has a lifetime which is half of the maximum connection time. Aging occurs when a filter has reached the end of its lifetime, which is five seconds in our example above. The aging process moves the newer filter to the older position (which can be as simple as updating a pointer), and clears the older filter and moves it to the newer position.

The aging process is as follows, and is repeated once for each direction;

1) The older filter is cleared. Any flow information that was in this filter is lost.
2) The newer filter is aged to become the older filter, possibly by simply updating a pointer
3) The older filter is recycled to become the newer filter.

As a result of the dual-filter setup, flows that are received just before the aging process takes place will be moved to the older filter. Once in the older filter the connection state will be maintained until the next aging occurs, therefore the lower bound of the maximum connection time is increased to half the maximum connection time (5 seconds in our example).

IV. Experimental Results

This section describes the expected performance of BDFT and SCD when deployed in real-world situations. Internet traffic traces are used to drive the experimental implementations of BDFT and SCD to obtain an indicator of expected performance. Analysis and validation was accomplished by implementing SCD and running experiments based on Internet traffic traces. To verify the SCD results, a 100% accurate flow tracker tracks the true connection status of every flow observed in the traces. This flow tracker was implemented using standard per-flow tracking and measurement techniques. We defined a flow to be the standard 5-tuple of IP source and destination address, TCP source and destination port, and protocol type.

To evaluate the results of our experiments the connection status reported from the flow tracker was compared to the results returned from SCD, giving one of three results. First the SCD algorithm can indicate that the flow is established, in agreement with the flow tracker, this counts as a successful indication by SCD. Second, SCD returns that the flow is established, but the flow tracker indicates that the flow is not established, this is a false positive and is a characteristic of bloom filters as explained in section III. Third, SCD can fail to report an established flow, and the flow tracker indicates that the flow is established, this is a false negative and occurs when the flow took longer to establish than the maximum flow connection time parameter of SCD.

The SCD algorithm was implemented in the C programming language using standard techniques of bitmap manipulation to implement the Bloom filters. Packets are read from the trace and the Bloom filter hashes are generated by three independent hash functions from the following packet header information; IP source and destination address, and TCP source and destination port. The Dual-filter SCD method, described in section III-A, was implemented by clearing the old filter for either direction and then simply updating pointers to exchange the new and old filters. For every SYN packet in the trace the hashes are passed to the SCD_Packet function, which returns true if the flow is now established and false if it is not. The connection establishment indication is stored and compared with the final state of the flow to determine if it was a successful indication.

We obtained traces of Internet traffic from the well-known networking research organizations CAIDA [20] and NLANR [11], with the two traces hereafter referred to as *C_04* (CAIDA) and *N_12* (NLANR). This section describes many of the intrinsic characteristics of these two traces and explains why they are representative of the diverse extremes of Internet traffic. In general, the *C_04* trace represents normal "dirty" public backbone Internet traffic, with many packets being invalid attempts at port scanning or DDoS attacks. This trace was collected by CAIDA from both directions of an OC48 link at AMES Internet Exchange (AIX) on Apr. 24, 2003, at Mountain View, CA, a west coast peering link for a large ISP [20]. The second trace, *N_12*, was obtained by NLANR in December of 2003, from their NCAR Gigabit tap (at the National Center for Atmospheric Research, Boulder) [11]. This trace represents the other end of the traffic spectrum from *C_04*, being fairly "clean" and containing a low number of active flows and very little or no attack and port scanning traffic.

For our simulations two one-hour long traces were selected which are representative of classic Internet traffic mixes. We obtained the traffic traces from two well-known networking research organizations; CAIDA [20] and NLANR [11], with the two traces hereafter referred to as *C_04* (CAIDA) and *N_12* (NLANR). This section describes many of the intrinsic characteristics of these two traces and explains why they are representative of the diverse extremes of Internet traffic. In general, the *C_04* trace represents normal "dirty" public backbone Internet traffic, with many packets being invalid attempts at port scanning or DDoS attacks. This trace was collected by CAIDA from both directions of an OC48 link at AMES Internet Exchange (AIX) on Apr. 24, 2003, at Mountain View, CA, a west coast peering link for a large ISP [20]. The second trace, *N_12*, was obtained by NLANR in December of 2003, from their NCAR Gigabit tap (at the National Center for Atmospheric Research, Boulder) [11]. This trace represents the other end of the traffic spectrum from *C_04*, being fairly "clean" and containing a low number of active flows and very little or no attack and port scanning traffic.

Table I describes the characteristics of the traces we used. Out of 12.7 million flows in the *C_04* trace only 556,000, or 4.3%, are valid fully-established flows. The high invalid-flow ratio combined with the high number of active flows make this trace a good worst-case test of SCD. The second trace, *N_12*, represents the other end of the traffic spectrum, containing a low number of active traces and very little or no attack and port scanning traffic. Out of 358,048 flows in this trace 274,473 are valid (76.7%), making this trace an example of the best-case performance of SCD.

Table I also contains two additional columns. The "SYN, FIN, RST" column shows the percentage of packets from the trace that are TCP packets with the SYN, FIN, or RST flags on. Average Active Flows gives a rough estimate of the number of flows actively transmitting data in the trace. Flow timeouts are set to thirty seconds for flows that have contained at least one FIN or RST, and ten minutes for all other flows. Flows that began before the start of the trace, or have their connection phase span the end of the trace, are ignored by the flow tracker and are not included in the results.

Our experimental results are presented in Figure 2, Figure 3, and Table II. To evaluate the performance of SCD, we varied two parameters of the algorithm, filter size and maximum connection time. Filter size represents the size of one Bloom filter in bits. Given that dual-filter SCD uses four filters, the total memory usage can be calculated as:

$$\text{Memory Usage} = \left(\frac{\text{bits per filter}}{8 \text{ bits/byte}} \right) * 4 \qquad (1)$$

The maximum connection time specifies the maximum time that can elapse between a SYN packet in one direction and

TABLE I
TRACE CHARACTERISTICS

$TraceName$	$TotalPackets$	$TotalTCPFlows$	$AverageActiveFlows$	SYN, FIN, RST
CAIDA 2003-04 (C_04)	202510985	12795449	901245	10.71%
NLANR 2003-12 (N_12)	196960126	358048	11284	0.66%

TABLE II
SCD PARAMETERS AND ACCURACY

$Trace$	$Memory(bytes)$	$Max.Conn.Time$	$Accuracy$
N_12	524288	64	99.9996%
N_12	32768	16	99.9982%
C_04	524288	32	99.9685%
C_04	32768	8	99.0553%

TABLE III
COMPUTATIONAL EFFICIENCY

$\%time$	$selfsec$	$calls$	$name$
36.94	1105.5	161273588	FindFlow
18.75	561.12	322547176	StoreComps_o
14.19	424.76	330	AgeFlows
9.94	297.61	12225610	RemoveFlow
3.2	95.63	71668476	**HashLookup**
3.02	90.39	186368015	StoreFlow
1.06	31.85	41426043	**HashSet**
0.87	25.92	186368015	**SCD_Frame**
0.63	18.98	11798670	NewFlow
0.2	5.84	13808681	**HashAdd**
0.17	5.13	12322186	**PacketHash_rev**
0.15	4.53	11229407	**PacketHash_fwd**
0.11	3.39	14359349	**SCD_Packet**
0.09	2.72	12225610	ProcessFlowEnd

the response SYN from the other direction. Due to the nature of dual-filter SCD, the actual maximum connection time on a per-flow basis varies from Max Conn Time / 2 to Max Conn Time. If this time is exceeded a false negative is generated.

Table II lists example configurations and shows that SCD is 99%+ accurate even at only 32k bytes of memory usage. This level of memory usage indicates that SCD can be implemented using SRAM at the datapath level of a router or other network device. By increasing memory usage to 512k, over 99.9% accuracy can be achieved. As a comparison of memory usage, our per-flow tracker used 24MB of memory to store about one million flows, or forty-eight times more memory than the 99.9% accurate SCD.

An important observation can be made by analyzing the SCD results. Some of memory usage plots in Figure 2 have a local minimum, indicating that there is an ideal setting for the maximum connection time parameter. The ideal setting occurs when the false negative and false positive rates balance out. With a low maximum connection time setting many flows fail to establish before the filters are cleared, leading to a high number of false negatives. As the maximum connection time parameter increases, the Bloom filters start to become overloaded with flows, leading to a high number of false positives. The increase in false positives is balanced by the decrease the number of false negatives as maximum connection time increases. False negatives decrease for two reasons. First, the direct reduction; flows that take a long time to establish may take less than the new maximum connection time, resulting in a false negative turning into a successful result. The second, is less obvious; the increasing number of false positives from Bloom filter overloading result in some beneficial errors; flows that still exceed the maximum connection time, and therefore should be false negatives, are reported as being connected due to Bloom filter errors.

A. Computational Performance

The computational requirements of SCD are determined by the specific hardware that the algorithm will be executed on. Scaling to 10+Gbps speeds requires use of SRAM for per-packet memory accesses, and an Application Specific Integrated Circuit (ASIC) or Network Processing Unit (NPU) for execution. To ensure scalability to 10+Gbps requires a hardware implementation of Bloom filters in the ASIC, or a Bloom filter program in the NPU. Low power hardware implementations of Bloom filters have been shown to be viable in several papers [12] [18].

To get a rough idea of the computational requirements and efficiency of SCD we ran our experiment program through the Linux profiler gprof (for trace C_04). Table III shows an abbreviated portion of the gprof output. The % time and self seconds columns represent the length of time spent in the function. The functions FindFlow, AgeFlows, RemoveFlow, and StoreFlow represent the computational requirements of our flow tracker, and the functions in bold represent SCD. To decrease the lookup time in the naive flow tracker, flow lookup was implemented using a one million element hash table, effectively reducing the number of lookup operations per packet to one, given that there are less than one million active flows. However, even with this drastic attempt to increase the efficiency of the flow tracker the lookup time still accounts for 37% of execution time, and in total the naive flow tracking functions account for 64%. By comparison, the performance advantage of an SCD implementation is clear. Only 5.76% of execution time, or 172.29 seconds (which is the sum of the SCD execution times, shown in bold) were required to process the 3600 second trace, leading to a 11x reduction in processing requirements.

In addition to the intrinsic efficiency of SCD, note that in typical network hardware many of the functions of SCD are implemented in hardware, such as the hash calculations,

and bit-wise manipulation and addressing. If these functions were implemented in hardware it would essentially remove the PacketHash functions, and much of the processing time in HashLookup, HashSet, and HashAdd.

V. Concluding Remarks

Increases in processing requirements for network measurement applications will continue for the foreseeable future, as the required statistics become more complex. This increase combined with the large number of fake flows in Internet traffic, due to DoS attacks and port scanning, cause network measurement applications to be overloaded processing traffic that is not of interest to the operator. Processing-intensive measurement applications can benefit greatly from the up to 95% reduction in flow records that SCD can provide.

We have shown that SCD provides a viable real-time method of reporting fully established TCP flows. Using very little memory, SCD is able to achieve accuracy of 99%+. In addition, SCD can be implemented using hardware-based bloom filters or on network processors that use SRAM memory. The parameters of SCD are flexible and only need to be set to approximately the ideal value to achieve high accuracy. Also note that as an area of future work it may be possible that with slight modifications SCD can be used for port scan detection and detection of some attacks. The bloom filters can be modified to counting bloom filters, and the hashes can be based on IP addresses only. In this way it would be possible to track the number of failed connection attempts on a per-IP basis, with some errors.

VI. Acknowledgments

Support for this research was provided by the Center for Communications and Information Technology, a division of Ontario Centers of Excellence, and Alcatel-Lucent. The experimental results were generated from traces made available to us by CAIDA and NLANR.

References

[1] Snort. `http://www.snort.org/` (last accessed on Dec 16, 2006).

[2] Michael E. Attig and John Lockwood. SIFT: Snort intrusion filter for TCP. In *Symposium on High Performance Interconnects*, pages 121–127, Aug 2005.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: a survey. *Internet Mathematics*, 1(4):485–509, 2004.

[5] N. Brownlee, C. Mills, and G.Ruth. Traffic flow measurement: architecture (RFC 2722),. `http://www.faqs.org/rfcs/rfc2722.html` (last accessed on Dec 17, 2006).

[6] Loris Degioanni and Gianluca Varenni. Introducing scalability in network measurement: toward 10 gbps with commodity hardware. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 233–238, 2004.

[7] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 325–336, 2003.

[8] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. *SIGCOMM Computer Communication Review*, 34(4):245–256, 2004.

[9] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 323–336, 2002.

[10] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 153–166, 2003.

[11] National Laboratory for Applied Network Research. NCAR-1 trace, collected in December, 2003, NSF ANI-0129677 (2002) and ANI-9807479 (1998),. `http://pma.nlanr.net/Special/ncar1.html/` (last accessed on Dec 17, 2006).

[12] T. Kocak and I. Kaya. Low-power Bloom filter architecture for deep packet inspection. *Communications Letters, IEEE*, 10(3):210–212, Mar. 2006.

[13] Abhishek Kumar, Jun (Jim) Xu, Li Li, and Jia Wang. Space-code Bloom filter for efficient traffic flow measurement. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet measurement*, pages 167–172, 2003.

[14] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[15] DARPA Internet program. Transmission control protocol,. `http://www.faqs.org/rfcs/rfc793.html` (last accessed on Dec 17, 2006).

[16] J. Quittek, T. Zseby, B. Claise, and S. Zander. Netflow version 9,. Cisco Systems `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html` (last accessed on Dec 16, 2006).

[17] Darren Reed. Ipfilter. `http://coombs.anu.edu.au/~avalon/` (last accessed on Dec 16, 2006).

[18] Elham Safi, Andreas Moshovos, and Andreas Veneris. L-cbf: a low-power, fast counting bloom filter architecture. In *ISLPED '06: Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, pages 250–255, 2006.

[19] K. Shah, S. Bohacek, and A. Broido. Feasibility of detecting TCP-SYN scanning at a backbone router. In *Proceedings of the American Control Conference*, pages 988–995, Jul 2004.

[20] Colleen Shannon, Emile Aben, kc claffy, Dan Andersen, and Nevil Brownlee. The CAIDA OC-48 traces dataset, collected in April, 2003,. `http://www.caida.org/data/passive/` (last accessed on Dec 17, 2006).

[21] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th USENIX Security Symposium*, pages 29–44, Aug 2004.