COTraSE:
Connection Oriented Traceback in Switched Ethernet

Marios Andreou, Aad van Moorsel
Newcastle University, School of Computing Science
{M.S.Andreou}{Aad.vanMoorsel}@ncl.ac.uk

Abstract

Layer 2 traceback is an important component of end-to-end packet traceback. Whilst IP traceback identifies the origin network, L2 traceback extends the process to provide a more fine-grained result. Other known proposals have exposed the difficulties of L2 traceback in switched ethernet. We build on our earlier work and improve in a number of dimensions. Memory requirements are decreased by maintaining 'connection records' rather than logging all frames. Our switchport resolution algorithm provides error detection by correlating MAC address table values from two adjacent switches. Our solution also takes stock of potential transformations to packet data as this leaves the local network. We have implemented the core algorithm and used data from available WAN traces to demonstrate the potential memory efficiency of our approach.

1. Introduction

In receiving an IP packet, the header source address is often taken as an indication of the originating machine's identity. However, the Internet Protocol does not prevent creation of packets with *forged* source address. Explicit generation of these 'spoofed' packets requires a degree of planning and effort and so they are often associated with malevolent network activities [6]. The most widely known of these are Denial of Service attacks; it is obvious for instance that obscuring the origin of an attack may prolong its effects.

Another form of 'spoofing' that is typically transparent to the user results from commonly employed resource provisioning mechanisms. In a Local Area Network (LAN), a number of machines may share a smaller number of public IP addresses through the use of Network Address Translation (NAT), and repetitive requests for commonly accessed web pages are minimized with a proxy web cache. This results in the source address of packets being *overwritten* with that of a gateway router.

Systems that aim to identify a given IP packet's originating machine, regardless of *forged or overwritten* source address are termed 'IP Traceback', and a number of proposals exist in this area [1, 2, 3, 4, 5]. A common shortcoming, however, is the trace result's granularity. IP Traceback reveals the origin *network*, but not the origin *host* (i.e., at best one identifies the origin's first hop router [7]).

A sub-domain of IP Traceback has emerged in recent years to counter this shortcoming, with proposals for "Layer 2 Traceback". Generally, these 'internal' traceback systems extend the tracking process beyond the leaf router, and into the originating network [7, 8, 9]. Known approaches use a hashed digest to represent each packet. They do not, however, take into account potential transformations to packet data as this exits the local network (e.g., due to NAT). Thus, it is not possible to map the IP data delivered to the intended recipient, with (the hashed digest of) that data *prior* to transformation.

As a result, it is not possible to service traceback requests from non local hosts, and so the utility of such systems in real world scenarios is somewhat limited. Even if the eventual wide-spread adoption of IPv6 renders NAT obsolete, mechanisms such as web-caching will still be a necessary part of a typical LAN deployment (especially so for larger corporate/university networks).

Insights provided by our earlier work [9] enabled the development of 'COTraSE', a 'connection-oriented' logging based traceback system for Switched Ethernet (*swEthernet*). COTraSE decreases storage requirements whilst maintaining the requirement of *accountability for any given packet*. Uniquely, we correlate MAC address table (MAC-table) entries from two adjacent switches to establish causality between a MAC-address and the origin switchport. This correlation allows us to identify a number of potentially malevolent conditions (explained later).

Furthermore, COTraSE makes no assumptions about network topology, and provides for any potential transformations to the IP header data as this leaves the LAN. An ideal deployment providing trace results with the highest granularity requires logging between all network switches, however COTraSE accommodates partial deployment with reasonable results. We have implemented the core algorithm which takes as input anonymised WAN traces from [10, 11, 12], to enable discussion of memory requirements given in Section 4.

COTraSE was in part motivated by EU legislation that requires the retention of data in communications networks [14]. Such a system requires local access to network data if the identity of the originating machine is to be *reliably* attached to that data. Layer 2 Traceback (L2 Traceback) systems typically treat frame MAC addresses as 'guilty ́till proven innocent' as they are easily spoofed. We feel that *L2 Traceback* systems can provide the requisite 'connection records' identified in the aforementioned EU Council Directive.

The rest of this paper is as follows: Section 2 contains related work, and the main challenges exposed by those. In Section 3 we present COTraSE and its key algorithms. Section 4 addresses our Java implementation and the WAN traces used as input, to end with a discussion of memory requirements. Section 5 considers deployment issues and we conclude in Section 6.

2. Background

Determining the identity of an attacker may require (up to) three stages [8]. Stage 1, *IP traceback*, reveals the origin network and stage 2, *Layer 2 Traceback* (L2 Traceback) reveals the origin host. In the case of *overwritten* addresses, stage 1 *is not necessary*, as the network is identified by default. Finally, if stage 2 reveals a 'zombie host' then stage 3, 'Connection (chain) traceback' [13] may identify the true origin. COTraSE is an L2 Traceback system as described in Section 3.

Generally, L2 traceback systems combine *Switch Identifier (sID)* and *Switchport Number (pNO)* to uniquely identify each host on the L2 network. They borrow from *IP Traceback* in their overall approach, that is, *packet marking, messaging* or *logging*.

Link-Layer Traceback in Ethernet Networks [8]

'Tagged Frame Traceback' (TRACK) is unique in using a hybrid approach, with both packet marking *and* logging elements. Tags are applied to ethernet frames by an 'in switch' process. Each tag includes a keyed-Hash Message Authentication Code over the first 32 bytes of IP data carried by its frame. A separate process in the 'Analysis and Collection Host' removes and logs the tags with links to a sorted 'host table' (each host is $sID + pNO$).

This very interesting proposal trivialises a core process of other known solutions: establishing causality between a given frame and the originating switchport (pNO). TRACK assumes an 'in switch' process, and we agree that this *is* the best vantage point for establishing pNO. However, implementing traceback 'in switch' assumes functionality that is (typically) not available.

Layer-2 Extension to Hash-Based IP Traceback [7]

Hazeyama *et al* are the first to adapt the Source Path Isolation Engine (SPIE) for swEthernet. SPIE is a logging IP traceback system where a 'Data Generation Agent' (*DGA*) logs a hashed digest of each packet forwarded by a router [1, 15] using bloom filters to achieve significant memory efficiency.

In [7] the authors deploy their extended DGA, *xDGA*, within gateway routers. For each received frame, *sID* is inferred from the frame *destination* MAC address (i.e., that of the recipient router interface). By maintaining a local copy of each switch MAC address table (MAC-table) and given *sID*, the appropriate MAC-table is used to obtain the port number (*pNO*) based on the frame *source* MAC address. The *sID* and *pNO* are included in the digest input.

Using a single bloom filter for all packets makes traceback request processing expensive. More significantly, a specific topology is *required* where all hosts are separated from the network 'edge' by only a single switch, as otherwise sID cant be inferred from destination MAC addresses.

Logging Based IP Traceback in Switched Ethernets [9]

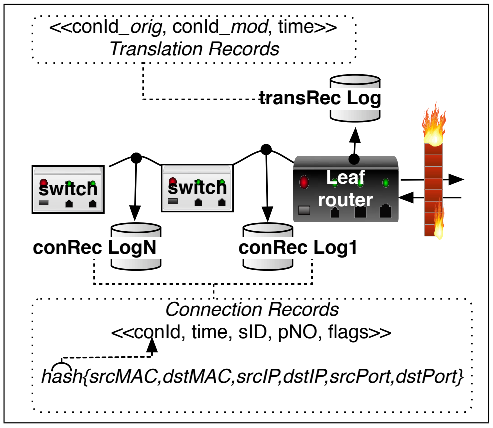
Our earlier work [9] is a second adaptation of SPIE for swEthernet, where we addressed difficulties encountered by the proposal above. We log *at each switch* with a tap-box running our *switch-DGA*, which receives traffic from 'port mirroring'. *Switch-DGA* uses a bloom filter array, with an element for each switch port. The hash output is stored in the bloom filter representing the origin port, established from the local switch MAC-table. When querying archived bloom filters the given array index reveals the source pNO.

A major problem encountered was that though the switch could reliably mirror traffic with little affect to the 'primary' switching functions, the port and host receiving mirrored traffic were quickly overwhelmed. More significantly, we also realised that none of the known approaches to L2 traceback (including our own) considered the effects of NAT and other such processes, as mentioned earlier.

3. COTraSE

In COTraSE, rather than switchport mirroring we instead log *between* switches with a passive tap. Memory requirements are decreased by attributing frames to 'connections' and logging representative 'connection records' (*conRecs*) for each interval. We note that a similar approach is used in router-deployed NetFlow [16], which is tailored to performance management, not traceback.

Packets exchanged between peer network processes at a given time will have the same source and destination MAC, IP and TCP/UDP addresses, and this data is used as the connection identifier (*conId*). One can group packets in 'connections' even in the absence of TCP/UDP at the transport layer, as we will see. A separate process at the network edge maintains 'translation records' (*transRecs*), addressing the issue of overwritten data (as in NAT). Figure 1 gives an overview of the COTraSE system.



<<conId_*orig*, conId_*mod*, time>>
Translation Records
transRec Log
Switch
Switch
Leaf router
conRec LogN
conRec Log1
Connection Records
<<conId, time, sID, pNO, flags>>
hash{srcMAC,dstMAC,srcIP,dstIP,srcPort,dstPort}

Figure 1. COTraSE - *conRec* and *transRec* Logs

3.1. The conRec Logs

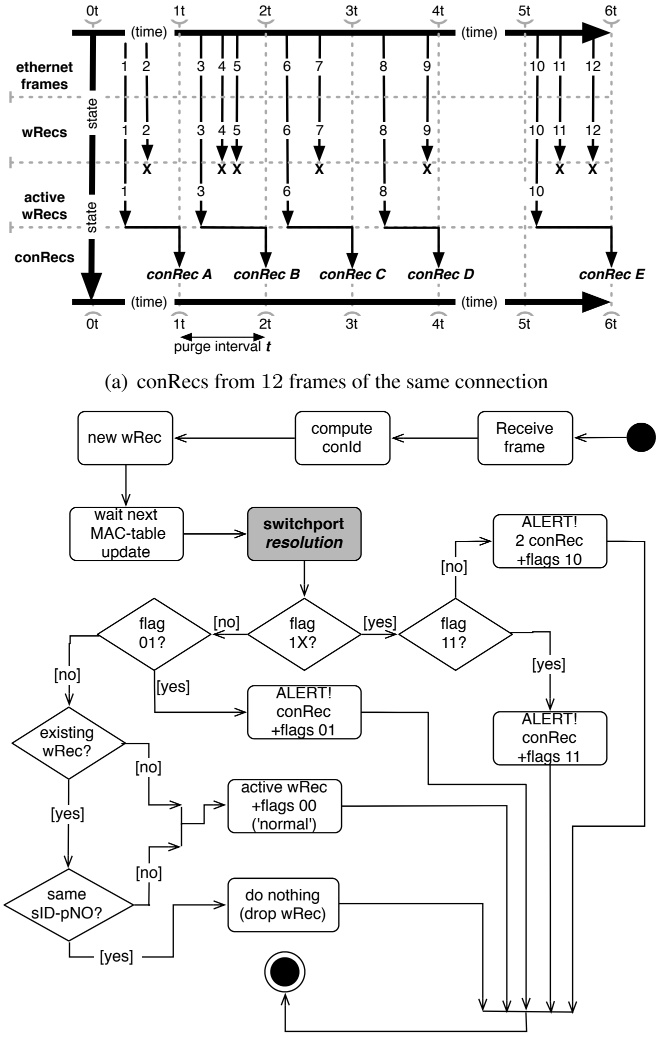
Each connection record (*conRec*) consists of connection identifier (*conId*), timing and origin information, which is switch identifier *sID* and port number *pNO*.

We use a cryptographic hash function over the data-link, network and transport layer addresses (MAC, IP, TCP/UDP data) to derive the connection identifier (*conId*) for each received frame. These addresses can uniquely identify a specific connection between two peer processes *only if* they are taken together.

Thus, for non TCP/UDP traffic and in the absence of ports, deriving *conId* requires us to find a different means of grouping frames. Source and destination MAC addresses are used in conjunction with other available data. This is protocol dependent and we give two characteristic examples: for *ARP* one can use the *Sender* and *Target* Protocol Address together with the *Opcode*, whilst for *ICMP* the IP source and destination addresses are available, and used in conjunction with ICMP *Type* and *Code*.

For each frame received at a conRec log, *conId* is computed and used to make a new working record (*wRec*). After *switchport resolution* (explained subsequently), each wRec becomes 'active' *only if* there is no existing *active wRec* with the same conId. Thus at a given time only one active wRec will exist for each connection.

The active wRecs are cleared as per the 'active connections purge interval' and moved to permanent storage as *conRecs*. The *next* frame for each connection will then become the new active wRec. A conceptual overview of this process is given in Figure 2(a), showing 12 frames from the same connection. During each purge interval only *one* wRec becomes active, whilst the rest are discarded.



(a) conRecs from 12 frames of the same connection

(b) conRec log algorithm

Figure 2. The conRec Log

Figure 2(b) gives the conRec Log algorithm. As can be seen, once conId is computed and a new wRec created, switchport resolution is deferred until the next *MAC-table update*. This is when the conRec log's *local MAC-tables* are updated from the adjacent *switch MAC-tables*. It is possible that *local MAC-tables* do not yet contain an address, such as when a new host joins the network. Its likelihood is reduced by deferring switchport resolution until after the local MAC-tables are updated.

3.2. Switchport Resolution

As mentioned earlier the source MAC (*srcMAC*) address cannot be taken 'at face value' as it is easily spoofed. Switches 'learn' the $srcMAC \leftrightarrow pNO$ mapping from each forwarded frame and update their MAC-table accordingly. As in other proposals, we maintain local copies of the switch MAC-table to determine the *Switch Identifier (sID)* and *Switchport Number (pNO)* based on *srcMAC*. In CO-TraSE this process is termed 'switchport resolution'.

Our approach is to correlate the MAC-table values from *both* adjacent switches to determine sID and pNO. Each switchport is either an *access port* to end hosts or a *external link port* to another switch. The ports connecting the switch to the conRec Log are termed *internal link ports*.

This classification is configured at each conRec log, allowing the return of MAC-table lookups to be classed as one of: null, external link (*ext_link*), internal link (*int_link*), access port (*axs_port*). It is likely that the MAC-tables from both switches will contain an address mapping for a given address. Based on our classification, some combinations are not acceptable, and generally indicative of an error as will be explained below. One *legitimate* combination is for the (local) MAC-table of switch 'A' to report an int_link whilst the MAC-table for switch 'B' gives an axs_port.

In such cases, switchport resolution must 'decide' which of the two should be returned as the $sID + pNO$ identifier. An axs_port gives a specific network access point and so is always chosen if available. If both tables return links, then an ext_link is preferred over an int_link, as in the former case neither adjacent switch was the origin.

Figure 3 shows the switchport resolution algorithm. As can be seen, the algorithm makes use of a 2 bit flag to signal the outcome of MAC-table lookups. Code 00 indicates 'normal' switchport resolution, that is, a single axs_port or ext_link is returned. This is shown as cases $2, 8$ and $4, 6$ respectively, in Figure 3). Code 01 means a port mapping was *not* available for an address, corresponding to case 1. This does not necessarily imply an error; if MAC addresses are *consistently* not learnt by switch MAC-tables, then this may indicate an oversubscribed switch [17].

The wRecs with codes other than 00 are never made 'active', as can be seen in Figure 2(b). Depending on available information one or more conRecs are created, with the error codes indicating their status. Codes 10 and 11 signal 'erroneous' switchport resolution. In Figure 3 cases $5, 7, 9$ and 10 produce error 10 and case 3 produces error code 11. The former indicates that we cant 'choose' between conflicting or equivalent values. For instance, two ext_links

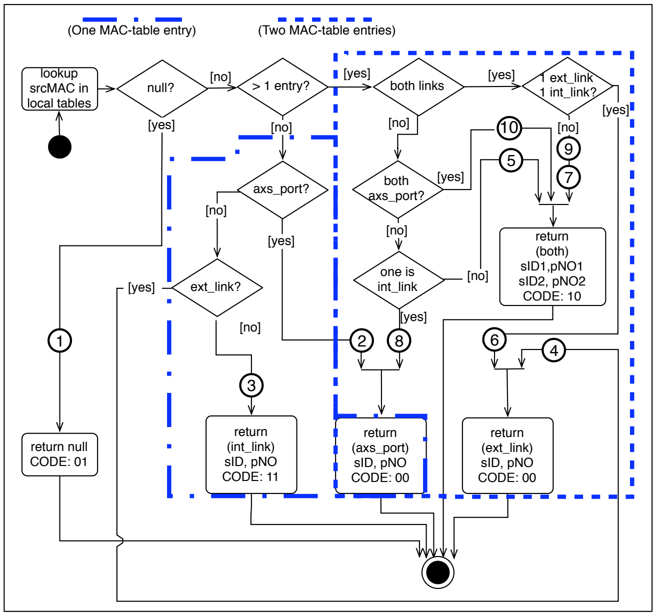


Figure 3. The switchport resolution algorithm

or two int_links would mean that the same MAC address was 'seen' as a source address from two opposing directions. This is impossible as switched Ethernet does not permit cycles[17].

Code 11 means there is *insufficient* information as only an int_link is returned. In this case, switch 'A' points to switch 'B' as the origin, through an int_link. Given that we perform switchport resolution after local MAC-tables are updated we expect switch 'B' to provide an axs_port or an ext_link.

3.3. Timing Parameters

The MAC-table update time must be such that a received frame is not 'aged out' from the switch MAC-table before local MAC-tables are updated. The IEEE suggest an aging time of between 10 and $1,000,000$ seconds [18] . Since unprocessed wRecs are buffered until the next update, memory utilisation is a practical concern. We choose a MAC-table update time of 5 seconds to stay within the suggested minimum. This creates a worst case requirement of $\approx$ 300Mbytes of RAM on a full duplex 1Gbit/s link to buffer unprocessed wRecs, as we will see.

The active connections purge interval (*purgeInt*) is governed by an inherent time/space tradeoff. A lower purgeInt means more frames become conRecs overall with each representing a smaller time interval. A higher purgeInt decreases conRec storage requirements, but also decreases time precision in traceback replies.

Traceback requests provide the time δ when a tracked frame was observed. Based on this we must decide which conRec 'represents' the requested frame. The purgeInt must be greater than the time we expect a frame to take in reaching its destination. With purgeInt t and round trip time x, the conRec for δ *or* the previous one 'represent' the tracked frame if $t > \frac{x}{2}$. This is because a frame received at the conRec log at $t + \alpha$ reaches its destination at $(t + \alpha) + \frac{x}{2}$. If $t > \frac{x}{2}$ then the frame will be delivered during the *current* or *next* purgeInt.

We choose 10 seconds as the *lowest* bound on purgeInt. By sampling data from Skitter [19] for 3 arbitrarily chosen days we determine that (at the $99th$ percentile) the mean round trip time is ≈ 3.5 seconds and we require $t > \frac{x}{2}$. We evaluate the impact of a 10 second purgeInt on memory requirements later.

3.4. The transRec Logs

At the transRec logs, we do not perform switchport resolution to determine sID or pNO. This is done at all conRec logs en route to the leaf router and doing it again provides no new information. An 'active transRec' set in maintained in a similar manner to the 'active wRecs' with the same factors governing its purge interval.

For each frame observed, *conId_orig* is computed as before. If there is no 'active' transRec for that connection based on conId_orig, then the connection's state is retrieved from the NAT or proxy web cache. This information is used as input to conId_mod and the transRec becomes 'active'. Note that conId_Mod does not include the source and destination MAC addresses.

For traceback requests from beyond the local network, transRecs are searched for *conId_mod*. If a match is found, a *local* traceback request is dispatched, and conRec logs queried based on *conId_orig*. Thus we 'traceback' *packets* regardless of address translation at the leaf router, but also *frames* that are (wholly) local.

4. Memory Requirements

Storage requirements are an important concern for logging traceback systems as they govern the 'traceback window'. This is the time during which we can traceback a packet after it has been logged (i.e., before older logs are overwritten). Below we give the COTraSE record formats (brackets denote size in bits):

> **wRecs (230):** conId (128), source MAC (48), sID (10), pNO (10), timestamp (32), flags (2)
> **conRecs (182):** conId (128), sID (10), pNO (10), timestamp (32), flags (2)
> **transRecs (288):** conId_orig (128), conId_mod (128), time (32)

With regards to 'operational' memory, taking a duplex link at 1Gbit/s, and with average frame size 1000 bits, *at most* one gets $\approx 1,000,000$ frames per second in either direction. Assuming a hypothetical worst case scenario where *all* frames belong to *different* connections, we require RAM of $\approx$ 300MBytes to store wRecs for 5 seconds, until the MAC-table update.

The number of connections C gives the lowest bound for the number of conRecs cR we need to store for each purge interval p. How many conRecs we store *overall* depends on the 'traceback window', in terms of number of purge intervals. Though C is not within out control, we can tune p, as explained earlier. A larger p will mean less purgeInts occur within a given traceback window and so less conRecs overall. The lowest bound for p, producing the greatest value for cR is 10 seconds, and this is used in our simulations.

We empirically deduce memory requirements against 'real' network traffic, a brief summary of which is given in Table 1.

Table 1. Trace data summary

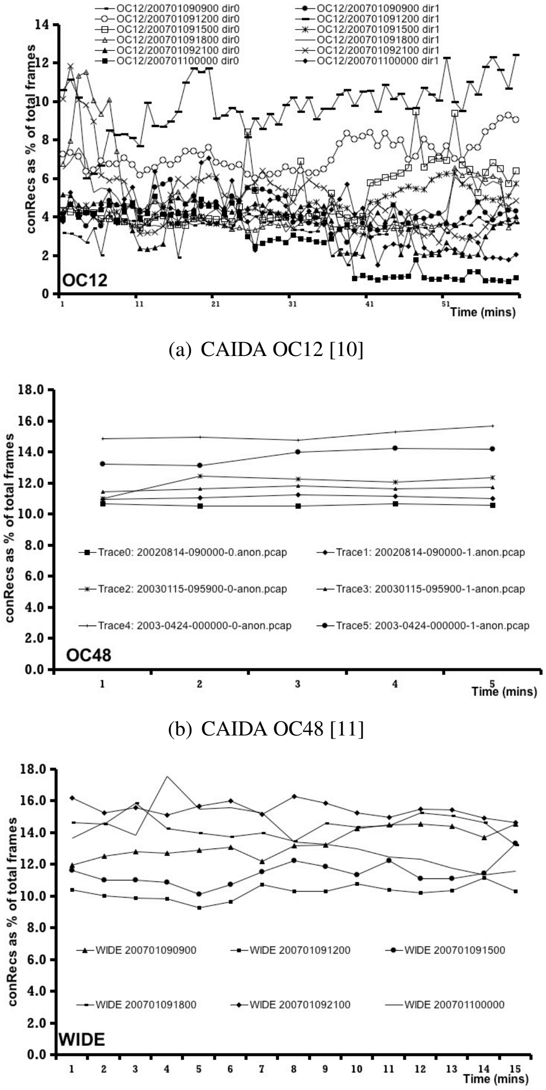
Source	Traces	Total Frames	Avg. Frames/min
OC12[10]	$12 * 60$mins	$136,803,068$	$189,588$
OC48[11]	$6 * 5$mins	$114,181,288$	$3,806,043$
WIDE[12]	$6 * 15$mins	$89,357,967$	$992,866$

Source 'OC12'[10] provides data from an OC12 link at the AMPATH Internet Exchange. We ran the conRec log with data from 09 Jan 2007, at 3-hourly intervals from 0900 until 0000 of 10 Jan. Source 'OC48' [11] is an OC48 peering link for a large ISP at AMES Internet Exchange. The data we use is from 14 Aug 2002 (0900), 15 Jan 2003 (0959) and 24 Apr 2003 (0000). Source 'WIDE'[12] is a 150Mbit/s (we assume OC3) transPacific link providing 15 minute traces for the same day (and times) as OC12.

We cannot draw *general* conclusions from these simulations especially as WAN traces are not representative of LAN traffic, which is the focus for COTraSE. However, 'real' LAN traffic is generally not widely available and, though frustrating, one can appreciate the associated privacy and security concerns.

Each trace was processed with 'tcpdump' to produce a hex dump of packet data complete with capture times. Our code then processed all TCP, UDP and ICMP data encountered which on average was more than 98% of the traffic. The wRecs created from 'received' frames were admitted to the active wRecs as explained earlier. An MD5 hash was computed for each packet's *conId* and the purge interval was set to 10 seconds.

At the end of each purgeInt, the 'active' wRecs were cleared and resulting set of conRecs written to file. In Figure 4 we demonstrate the reduction in memory requirements from logging conRecs rather than all frames, by plotting the number of conRecs as percentage of total frames in each minute. We see a similarity in the plots and especially so between Figures 4(b) and (c). A lower $\frac{conRecs}{frames}$ ratio signals a greater reduction in memory requirements compared to logging all frames.



(a) CAIDA OC12 [10]

(b) CAIDA OC48 [11]

(c) WIDE [12]

Figure 4. conRecs as a percentage of frames for each minute of the given trace

We use $\frac{conRecs}{frames}$ to approximate the number of connections. From Figure 4 we take the upper bound for $\frac{conRecs}{frames}$ at 15%. We assume the given link is at full utility and that on average each packet is ≈ 1000 bits. The average packet size in the OC48 and WIDE traces is actually ≈ 700*bytes*/packet and so our estimate is very conservative. In table 2 we give the predicted memory requirements of a COTraSE deployment, expressed in megabytes, for providing the given 'traceback window'. For example, in the OC48 case, taking 15% of 2 million packets per second, and multiplying by 182 bits per conRec gives 390Mbytes for a 1 minute 'traceback window'. The OC48 case is of particular interest as the $\approx 2,000,000$ packets/s throughput is similar to full utility of a duplex 1Gb/s ethernet link (1000 bit frames).

Table 2. Predicted COTraSE memory use, with each link at full utility

	packets/s	Megabytes (traceback window)	
Source	($\approx$)	1 min.	1 hour
OC12	$700,000$	137	$8,201$
OC48	$2,000,000$	390	$23,430$
WIDE	$165,000$	0.5	$1,933$

5. Deployment Issues

Performance: Maintaining conId as a hashed digest preserves user privacy, even when logs are compromised. However it is also computationally expensive, and its necessity depends on the intended use of COTraSE. As an L2 traceback system the privacy preserving hashed conId is preferred. However, the EU 'data retention' council directive [14] mentioned earlier requires that 'data be transmitted upon request'. With a hashed conId, we cannot for example extract connection data for any given source or destination IP address.

The second 'bottleneck' operation is the MAC-table lookup performed during switchport resolution. In *worst case* conditions a dedicated CAM might be a necessity in order to keep up with the packet load. This is a general problem for all logging based L2 traceback systems. Given that a switch already performs a MAC-table lookup for all frames, traceback functions would ideally be performed 'in switch' but as mentioned earlier, such functionality is currently not available.

Vulnerabilities: Buffering frames until the MAC-table update assumes that address mappings do not change between receipt of a frame and the update operation. An attacker may exploit this to spoof a tertiary host's address. The switch MAC-table would overwrite the mapping for the given address to identify the attacker's pNO as the origin. However, if the tertiary host transmits frames *after* the attacker but *before* the MAC-table update, then switchport resolution will mistakenly identify the tertiary host as the originator of the attack frame.

The attacker first must discover the tertiary host's MAC address which requires effort due to the low visibility of switched Ethernet. The attacker needs to predict when the tertiary host is transmitting, but also when the MAC-table update occurs. Furthermore, the MAC-table update time can be randomized (e.g., between 3 and 5 seconds) to make conditions even more adverse for the attacker.

Finally, an obvious attack on the COTraSE system is for an attacker to send a large number of frames designed to produce *different* conIds, to exhaust conRec log resources. We note that under these 'worst case' conditions our connection oriented approach will degenerate to no worse than an explicit frame log.

Partial Deployment: It may be prohibitively expensive to provide 'full' deployment with a conRec log between all network switches. However, we can adapt conRec log placement to the deployment network, though a partial deployment ultimately sacrifices 'local traffic visibility'.

A frame is processed by all conRecs encountered en route to the gateway router. However, only the conRec log adjacent to the frame's origin switch will be able to provide the 'axs_port' pNO which identifies the frame's origin switchport. All other conRec logs will 'point' to the origin within the network (i.e., by providing an ext_link pNO).

6. Conclusions

We introduced COTraSE a 'connection oriented' logging based layer 2 traceback system for Switched Ethernet. This allows traceback of ethernet frames that are wholly local but also for IP packets addressed to external recipients, regardless of any address translation mechanisms (e.g., NAT). Rather than explicitly logging all traffic we instead attribute frames to ongoing connections and log representative connection records in discrete intervals. Our switchport resolution algorithm establishes the origin switch and port for frames by correlating MAC address entries from both adjacent switches. This algorithm classifies the return of each table lookup to detect potential errors such as MAC address 'spoofing'.

We empirically demonstrated the potential memory efficiency of a 'connection oriented' traceback approach by simulating the conRec log algorithm using data from available WAN traces. Our simulations show a consistently low $\frac{conRec}{totalframes}$ ratio which is favourable in decreasing COTraSE memory requirements.

References

[1] Snoeren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Schwartz, B., Kent, S.T., Strayer, W.T.: *Single Packet IP Traceback*, IEEE/ACM Transactions on Networking, Vol. 10(6), (2002), 721-734

[2] Burch, H., Cheswick, B.: *Tracing Anonymous Packets to Their Approximate Source*, 14th System Administrator Conference (LISA), (2000), 319-327

[3] Savage, S., Wetherall, D., Karlin, A., Anderson, T.: *Network Support for IP Traceback*, IEEE/ACM Transactions on Networking, Vol. 9(3), (2001), 226-237

[4] Stone, R.: *Centertrack: An IP Overlay Network for Tracking DoS Floods*, 9th USENIX Security Symposium, (2000)

[5] Bellovin, S., Leech, M., Taylor, T.: *ICMP Traceback Messages*, Internet Draft, IETF, (2003)

[6] Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: *Internet Denial of Service (Attack and Defense Mechanisms)*, Prentice Hall, (2005)

[7] Hazeyama, H., Oe, M., Kadobayashi, Y.: *A Layer-2 Extension to Hash Based IP Traceback*, IEICE Transactions on Information and Systems, Vol. E86(11), (2003), 2325

[8] Snow, M., Park, J.: *Link-Layer Traceback in Ethernet Networks*, Workshop Proceedings, IEEE LANMAN, (2007), 182-187

[9] Andreou, M., van Moorsel, A.: *Logging Based IP Traceback in Switched Ethernets*, Proceedings 2008 ACM SIGOPS European Workshop on System Security, (2008)

[10] Shannon, C., Aben, E., Claffy, K.C., Anderson, D.: *The CAIDA Anonymized 2007 Internet Traces*, http://www.caida.org/data/passive/passive_2007_dataset.xml, CAIDA, (2007)

[11] CAIDA OC48 Trace Project: *CAIDA OC48 Traces 2002-08-14, 2003-01-15, 2003-04-24 (collection)*, http://www.caida.org/data/passive/index.xml#oc48, CAIDA, (2008)

[12] WIDE Project: *Packet traces from WIDE backbone*, MAWI Working Group Traffic Archive, http://mawi.wide.ad.jp/mawi/, WIDE, (2008)

[13] Almulhem, A., Traore, I.: *A Survey of Connection-Chains Detection Techniques*, 2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, (2007)

[14] European Parliament: *Council Directive 2006/24/EC of 15 March 2006 on the retention of data*, OJ L105, (March 2006), 54-63

[15] BBN Technologies: *Source Path Isolation Engine*, http://www.ir.bbn.com/projects/SPIE/spiehome.html, (2004)

[16] Claise, B. Ed.: *Cisco Systems NetFlow Services Export Version 9*, Network Working Group Request for Comments 3954 (Informational), http://www.ietf.org/rfc.html, (2004)

[17] Seifert, R.: *The Switch Book: The Complete Guide to LAN Switching Technology*, Wiley, (2000)

[18] IEEE Std. 802.1D-2004, Media Access Control (MAC) Bridges, IEEE, (2004)

[19] CAIDA: *skitter*, http://www.caida.org/tools/measurement/skitter/, CAIDA, (2006)