COTraSE
CONNECTION ORIENTED TRACEBACK IN SWITCHED ETHERNET

Marios S. Andreou and Aad van Moorsel
*School of Computing Science, Claremont Tower
Newcastle University, Newcastle upon Tyne, U.K. NE1 7RU
$[M.S.Andreou]$ $[Aad.vanMoorsel]$ @ncl.ac.uk

Abstract: Layer 2 Traceback is an important component of end-to-end packet traceback. Whilst IP Traceback identifies the origin network, Layer 2 Traceback extends the process to provide a more fine-grained result. Other known proposals have exposed the difficulties of Layer 2 Traceback in switched ethernet. We build on our earlier "switch-SPIE" 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.

Keywords: IP Traceback, Layer 2 Traceback, switched Ethernet traceback, network data retention, network accountability, EU 2006/24/EC.

I. 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 [1]. 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 IP address and TCP or UDP port numbers of packets being overwritten with those chosen by 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 great number of proposals exist in this area (see [2] through [24]). 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 [25]).
A sub-domain of IP Traceback has emerged in recent years to counter this shortcoming, with proposals for "Layer 2" traceback (L2 Traceback). Generally, these 'internal' traceback systems extend the tracking process beyond the leaf router, into the originating network [25, 26, 27], allowing the source of packets to be identified despite possibly spoofed MAC and IP addresses. Insights provided by our earlier work [27] enabled the development of COTraSE, a 'connection-oriented' logging based traceback system for Switched Ethernet. 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 switch-port. This "switchport resolution" allows us to identify a number of potentially malevolent conditions as we will see.
Furthermore, COTraSE makes no assumptions about network topology and provides for any potential transformations to the IP and TCP/UDP 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 by 'pointing' towards the origin host. We have implemented the core algorithm which takes as input anonymised WAN traces from [28, 29, 30], to enable discussion of memory requirements in Section 4.

Received June 10, 2009

Known L2 Traceback approaches do not take into account potential transformations to packet data as this exits the local network (e.g., due to NAT). 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 with similar consequences (such as web-caching) will still be a necessary part of a typical LAN deployment (especially so for larger corporate/university networks). COTraSE maintains translation logs at the exits of the local network to resolve issues arising from NAT and similar mechanisms.

COTraSE was in part motivated by EU Directive 2006/24/EC "on the retention of data ..." [31]. The core requirement of this European legislation is for providers of "publicly available electronic communications services" to maintain "communication records", akin to those maintained by providers of mobile telephony. We will see that the connection records created by COTraSE are a good match for the "communications records" required by Directive 2006/24/EC. Moreover we consider all work in the areas of logging based IP and L2 Traceback as directly relevant in exposing the difficulties faced by any implementation of this pan-European "data retention" system.

A preliminary version of this paper was presented at IAS 2008 [32]. In this article we outline our L2 Traceback system requirements and explain how COTraSE improves over our earlier switch-SPIE [27]. We provide supplementary details of the WAN trace data used by our implementation and expand on the calculation of COTraSE memory requirements. We also provide additional background material to aid the reader, including a discussion of the related Netflow system. In particular we consider how the flow expiration mechanisms adopted by Netflow differ from those of COTraSE and how this affects L2 Traceback.

The next section presents background on switched ethernet and related work in the area of L2 Traceback. Section 3 gives our L2 Traceback system requirements with a brief discussion of improvements made by COTraSE . Section 4 contains the details of the COTraSE system and Section 5 provides details of its implementation. Section 6 considers deployment issues and includes a discussion of Netflow. Finally we conclude in Section 7.

II. Background

Determining the source of an IP packet may require up to three stages [26]. In stage 1, IP Traceback can reveal the origin network and in stage 2, Layer 2 traceback reveals the origin host. In the case of overwritten addresses as in `NAT`, stage 1 is not necessary as the IP address can be assumed to be correct and the network is identified by default. Finally, if stage 2 reveals a 'zombie host' then stage 3, 'Connection (chain) traceback' [33] may identify the true origin. COTraSE is an L2 Traceback solution, thus targeting phase 2.

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.

Switched ethernet and L2 Traceback

A switched environment means that each host gets a dedicated link to the switch, and so medium access contention is eliminated. This bears upon the design of our L2 Traceback system in two ways, one of which positive and one negative. The negative consequence of a dedicated link is that there is no single vantage point from which to observe all traffic generated by the switch's connected hosts [34]. That is, each (unicast) ethernet frame is 'switched' from ingress to egress switchport and cannot be observed by a host on any other switchport. Our earlier work relied on switchport mirroring to overcome this issue [27]. In COTraSE we instead deploy logging at a tap between switches, as we will see.

The positive consequence of providing each host with an exclusive link is the need to create and maintain a MAC address table (MAC-table). This table associates the MAC address of each host with the switchport to which that host is connected. Thus, to correctly forward frames switches consult the MAC-table to determine the appropriate egress port. MAC-tables are constructed through a 'learning' process. A MAC address is reachable over the port from which traffic carrying that address as source was most recently seen. Each address can only be listed once; thus, if a host moves to another port (or the source MAC is forged), the MAC-table is updated. That is, ethernet switches submit all correctly received user data frames to the switch's 'Learning process' [35].

Each COTraSE logging node maintains local copies of the MAC-tables from its two adjacent switches to determine the origin switchport of received frames. Local MAC-tables are updated from the switch MAC-table at some predetermined rate, as we will explain.

Link-Layer Traceback in Ethernet Networks [26]

'Tagged Frame Traceback' (TRACK) [26] 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 [25]

Hazeyama *et al* [25] are the first to adapt the Source Path Isolation Engine (SPIE) for switched ethernet. SPIE is a logging IP Traceback system where a 'Data Generation Agent' (*DGA*) logs a hashed digest of each packet forwarded by a router [3, 36] using bloom filters to achieve significant memory efficiency.

In [25] 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* cannot be inferred from destination MAC addresses.

Logging Based IP Traceback in Switched Ethernets [27]

Our earlier work [27] is a second adaptation of SPIE for switched ethernet, 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.

III. Requirements and Contributions

COTraSE is a logging based Layer 2 traceback system designed for switched ethernet networks. Through switch-SPIE we improved over other L2 Traceback solutions by providing L2 Traceback independent of the topology of the switched Ethernet network. With COTraSE we further improve on switch-SPIE in terms of memory usage, by removing the logging bottleneck from the switches, by abandoning bloom filters (and its associated false positives) without incurring excessive storage costs, and by accounting for NATs and similar devices in our solution. We discuss these contributions in this section.

The generic requirements for any L2 Traceback system are as follows:

1. Identifies the origin switch ID and port number $\{sID, pNO\}$ of any given packet originating within the deployment network (within a bounded traceback window)
2. Requires no changes to existing network infrastructure or protocols.
3. Traceback enabling procedures (i.e. logging) should be considerate of user privacy.
4. The traceback result must be reliable.
5. Supports partial deployment and be applicable to any network topology.
6. Memory requirements at packet logs must be realistic.

With regards to identifying sID and pNO, COTraSE assumes that the source MAC address of all frames may be spoofed and so cannot be taken "at face value". COTraSE also allows for tracing despite "legitimate" spoofing, such as when Network Address Translation (NAT) is employed. This ability of COTraSE to traceback regardless of any legitimate spoofing is also relevant to requirement 2 by removing the conflict between logging based L2 Traceback systems and widely implemented mechanisms such as NAT.

COTraSE protects user privacy through encryption of logged data. This was also the case in switch-SPIE, where encryption additionally served the purpose of decreasing memory requirements. COTraSE does not rely on encryption to decrease memory requirements and instead adopts a "connection oriented" approach to logging. That is, the use of encryption is entirely for the preservation of user privacy. It could even be argued that the use of encryption in COTraSE is "overkill" as the data logged is drawn only from the headers of the Data-link, Network and Transport layers, with the payload not processed at all. In situations where performance of logging nodes is an issue then connection records may be stored in cleartext, avoiding costly cryptographic hashing routines.

Requirement 4, reliability of the trace result, is an area where COTraSE makes significant contributions. We feel that this is important in light of the intended use of data retained under EU directive 2006/24/EC. It must not be trivial to create traffic that implicates an innocent third party. COTraSE does not rely on bloom filters to decrease memory requirements and so false positives are eliminated alltogether. Furthermore, our switchport resolution algorithm (attributing a frame to a source switchport based on the source MAC address) makes use of two switch MAC tables to make conditions more adverse for an attacker. Finally, COTraSE does not rely on switchport mirroring to overcome the switched ethernet traffic visibility issue. The unreliability of port mirroring with increasing load was demonstrated by our experiments [27].

COTraSE offers increased support for partial deployment. In switch-SPIE we logged data from access ports, whilst switch link ports were not monitored at all. However, this is reversed in COTraSE where we exclusively log at switch link ports. An ideal deployment requires logging between every network switch. Only the logging node adjacent to a given frame's origin switch can provide us with the unique $\{sID, pNO\}$ to identify the origin host. However, given that we monitor link ports, a frame will be processed by all logging nodes en route to the gateway router. Thus, all logging nodes will 'point' towards the instigating origin within the local network. Thus, even in a partial deployment COTraSE can aid in identifying this host.

Finally, a decrease in memory requirements is another contribution of COTraSE. The aforementioned EU Directive 2006/24/EC requires that network data be retained for "periods of not less than six months and not more than two years". This is a much larger "traceback window" than that considered by any known L2 Traceback systems; in switch-SPIE for instance we discussed memory requirements in terms of minutes, not months. Thus, the efficiency of logging is a crucial factor in the real world utility of L2 Traceback systems. However, we will see that also using COTraSE it is more realistic to think in minutes than in days or months.

IV. COTraSE

The main idea behind COTraSE is that the series of packets exchanged to complete one interaction between peer network processes, will all have the same source and destination Data-link (MAC), Network (IP) and Transport layer (TCP or UDP) addresses. This data can therefore be used as a connection identifier (*conId*) for a given communication. (The term connection-oriented in COTraSE refers to this idea.) In so doing, memory requirements are decreased, attributing frames to connections and logging representative 'connection records' (*conRecs*) for each interval rather than explicitly logging all frames. It should be noted that a similar approach is used in router-deployed NetFlow [37], but NetFlow is tailored to performance management, and can therefore not be used directly for traceback. We will examine the differences between NetFlow and COTraSE in Section VI-A. Note furthermore that one can group packets in 'connections' even in the absence of TCP or UDP at the transport layer, as we will see in Section IV-A.

An overview of the COTraSE system is given in Figure 1. As can be seen, a *conRec* log is deployed between ethernet switches, processing traffic from the switch link ports. That is, rather than switchport mirroring as in switch-SPIE [27], COTraSE logs between switches with a passive 'tap'. This may be achieved using dedicated hardware, or with an inexpensive ethernet hub placed between the two switches. Though all ethernet frames are processed and undergo "switchport resolution" (see Section IV-B), only a subset are eventually logged as connection records. In addition, a separate process at the network edge maintains 'translation records' (*transRecs*), addressing the issue of legitimate spoofing (as in NAT). Note that depending on the length of the COTraSE purge interval (Section IV-A) there may be a number of *conRec* logs for each connection, but only a single *transRec* log is required and placed at the gateway router.

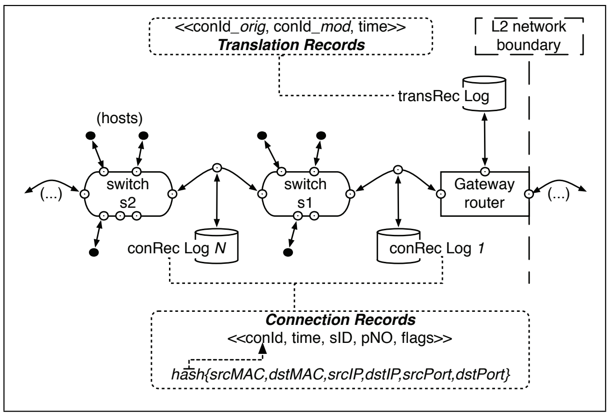


Figure. 1: COTraSE - *conRec* and *transRec* Logs

A. Connection Record Logs

Each connection record (*conRec*) consists of connection identifier (*conId*), timing and origin information. The origin host on the layer 2 network is identified by the unique pairing of switch identifier sID and switchport number pNO. To derive the connection identifier (*conId*) for each received frame, we use a cryptographic hash function over the data-link, network and transport layer addresses (`MAC, IP, TCP/UDP` data). Together, these addresses uniquely identify a specific communication between two peer processes.

For each frame received at a *conRec* log, *conId* is computed and used to make a new working record (*wRec*). After switchport resolution (explained in Section IV-B), each *wRec* becomes 'active' only if there is no existing active *wRec* with the same connection identifier. This means that at a given time only one active *wRec* will exist for each unique communication between peer network processes.

The active *wRecs* are cleared at the end of each 'purge interval'. To make this precise, let the set of active working records $\{^A wRecs\}$ represents all currently ongoing communications. At the end of each purge interval, for every active $w \in \{^A wRecs\}$ a new connection record is created and moved to permanent storage. In the next purge interval, the first frame for each connection will become the new active *wRec* and a new set of active working records is populated and maintained until the next purge interval.

A conceptual overview of this process is given in Figure 2, which depicts the progression from ethernet frame to connection record, for 12 frames of the same connection. The

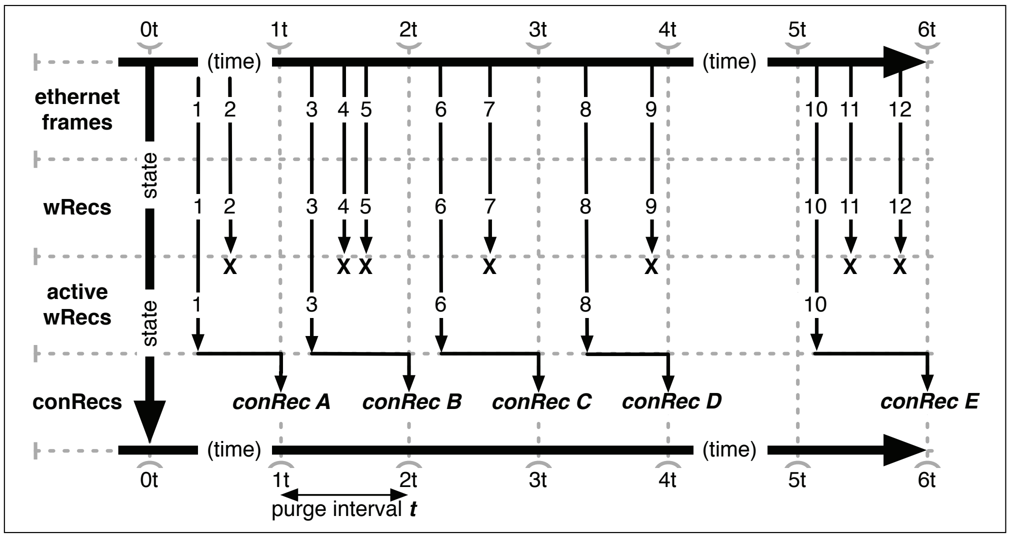


Figure. 2: *conRecs* from 12 frames of the same connection

horizontal axis shows time expressed in terms of purge intervals of length t. During the first interval $0t \leftrightarrow 1t$ two frames are received and a *wRec* created for both. However, only the *wRec* of frame 1 becomes active whilst the *wRec* of frame 2 is discarded (shown as **X**). After the first purge interval has elapsed ($1t$) the information contained within active *wRec* 1 is used to create *conRec A*. At the end of the depicted time period ($6t$) a total of five connection records are required to represent the 12 frames.

When processing traceback requests, COTraSE can provide the purge interval of length t during which a given frame was processed (if at all). There is an inherent space/time tradeoff in setting the length of the active connections purge interval. A larger interval means that more frames are represented by a single connection record, thus decreasing memory requirements. However this also means that identifying the time the packet was sent becomes less precise, since it can only be done with a precision equal to the length of the purge interval. We will examine the bounds of the purge interval in our discussion of timing parameters in Section IV-C.

For non `TCP/UDP` traffic and in the absence of Transport layer port numbers, deriving the *conId* requires us to find a different means of grouping consecutive frames from the same communication. Source and destination MAC addresses are used in conjunction with any other available data. The grouping of consecutive frames is protocol dependent and we give two characteristic examples as it would be impossible to provide an exhaustive list. 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`. We have been unable to identify a networking protocol for which some means of grouping is not possible. However, even in such a case the logging procedures will degenerate to no worse than an explicit frame log, which is the de facto modus operandi of all other known logging L2 Traceback systems.

B. Switchport Resolution

The *conRecs* themselves do not provide the unique switch port from which the frames originated. After all, the source MAC address may be spoofed. To establish the origin sID and pNO of a *conRec* , we maintain local copies of the switch MAC address table. This process of 'switchport resolution' is deferred until the next local MAC-table update. This is when the *conRec* log's local MAC-tables are synchronised with the adjacent switch MAC-tables. It is possible that the local MAC-tables do not yet contain an entry for the source address of a recently received frame, 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 as the switch MAC tables will contain an entry for all recently 'seen' source MAC addresses [34, 35].

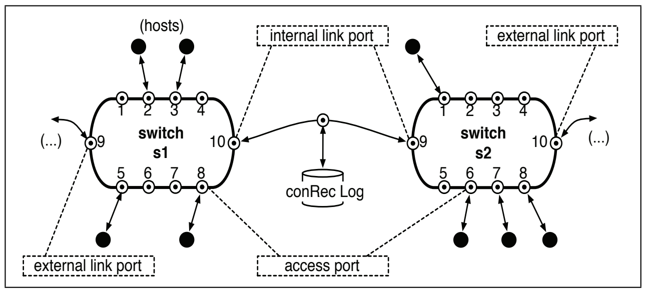


Figure. 3: COTraSE classifies all switchports as one of access, internal link or external link

Since we use a tap between two switches, we have the opportunity to correlate the MAC-table values from both adjacent switches to determine $\{sID, pNO\}$. We make use of this

fact to identify errors and possible malicious behaviour, in addition to determining the origin switchport. Each switchport is either an access port, providing network access to end hosts, or a link port which leads to another switch or router, and each link port is either an internal and external link. This is shown in Figure 3, where ports $\{s1, 10\}$ and $\{s2, 9\}$ are the internal link ports for the shown *conRec* log, whilst ports $\{s1, 9\}$ and $\{s2, 10\}$ are the external link ports. Of course this is relative to a specific connection record log; at the next *conRec* log beyond switch s2 (not shown), port $\{s2, 10\}$ would be an internal link port and port $\{s2, 9\}$ would be an external link port.
This classification of ports 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*).

Table 1: Summary of possible switchport resolution outcomes at a given *conRec* log

Case	Switch 1	Switch 2	Flags
1	null	null	01
2	null	axs_port	00
3	null	int_link	11
4	null	ext_link	00
5	ext_link	axs_port	10
6	ext_link	int_link	00
7	ext_link	ext_link	10
8	int_link	axs_port	00
9	int_link	int_link	10
10	axs_port	axs_port	10

Table 1 lists all possible outcomes of switchport resolution. The 2-bit flags are used to convey the outcome of switchport resolution. Flag 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. For instance, one legitimate combination is for the (local) MAC-table of switch $s1$ to report an int_link whilst the MAC-table for switch $s2$ gives an axs_port (case 8). Flag 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 [34].
The codes 10 and 11 signal 'erroneous' switchport resolution. In Figure 4 and Table 1 cases $5, 7, 9$ and 10 produce 10 and case 3 produces code 11. The former indicates that we cannot 'choose' between conflicting or equivalent values. For instance, two ext_links 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[34]. Flag 11, produced in case 3 means there is insufficient information as only an int_link is returned.
The actual resolution of the switch port pair $\{sID, pNO\}$ is based on Table 1 and follows the algorithm depicted in Figure 4. The algorithm uses the 2-bit flag to signal the outcome of MAC-table lookups (corresponding to the Flags column in Table 1). The numbered cases in Figure 4 match the cases in Table 1. The switchport resolution algorithm of Figure 4 returns the flag as well as the origin $\{sID, pNO\}$.

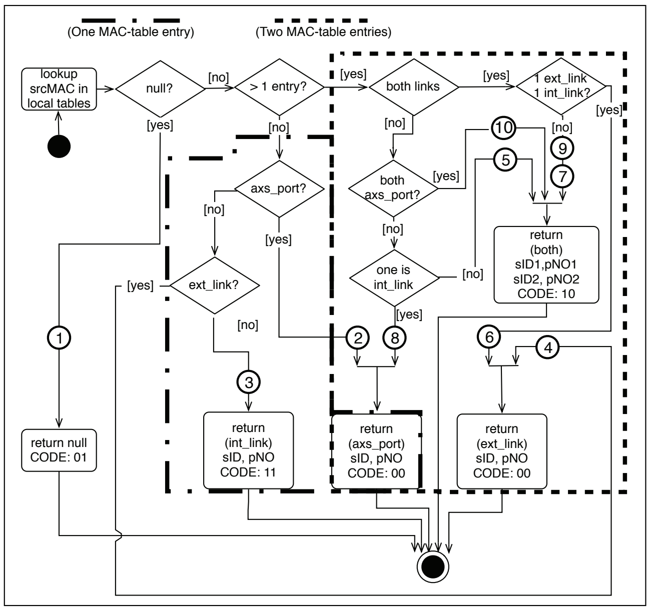


Figure. 4: The switchport resolution algorithm

When both switches return a non-null value (cases 5 to 10), switchport resolution determines which of the two should be used 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.
Note that the switchport resolution algorithm also returns a $\{sID, pNO\}$ identifier for the erroneous cases (with flag not equal to 00). It is then up to the integrated *conRec* Log and switchport algorithm to take the correct action. This is depicted in Figure 5. It depicts the complete interaction between switchport resolution and the *conRec* Log algorithm. At the start, when a frame is received the connection identifier is computed and a new working record created, according to Section IV-A. Periodically, the local copies of the MAC tables are synchronised with the adjacent switch MAC-tables (see Section IV-C for a discussion of the MAC table update frequency). The MAC table update triggers the switchport resolution algorithm, shown as a shaded activity box.
Depending on the outcome of switchport resolution each given working record may be dropped or made active, and in some cases the *wRec* becomes a *conRec* immediately, as can be seen in Figure 5. This last case is indicative of an error as uniquely COTraSE uses switchport resolution to detect potential sender spoofing activity. The *wRecs* with flags other than 00 are never made 'active', as was seen in Figure

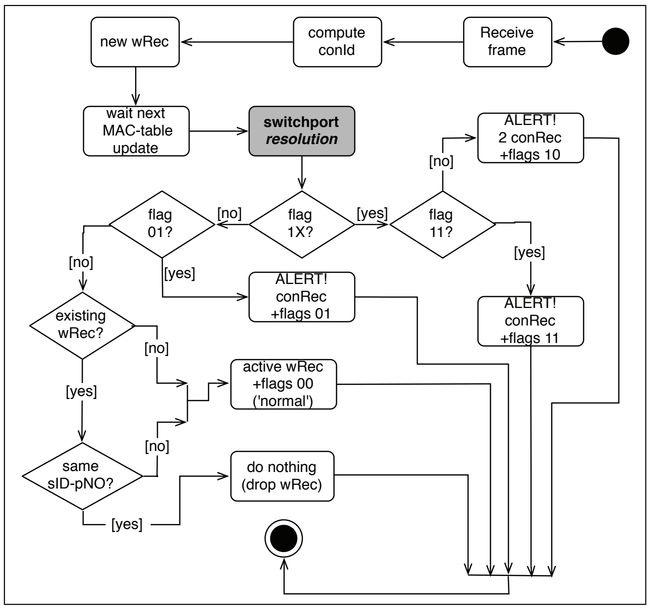


Figure. 5: Activity diagram showing the conRec Log algorithm

5. Depending on available information one or more *conRecs* are immediately created, with the flag indicating their status.

C. Timing Parameters

We consider two timing parameters that need to be set. First, we discuss the frequency of MAC table updates, and then we discuss the length of the purge interval.

MAC table update frequency. The MAC-table update time dictates when the local MAC-tables maintained at the *conRec* logs are refreshed from their corresponding 'master' tables on each of the two adjacent switches. As was shown in Figure 5 once an ethernet frame is received, its resulting working record is buffered to undergo switchport resolution only after the MAC-table update. Thus, we must ensure that the update time is less than the MAC address table 'aging time'.

All switch MAC-tables "age out" entries for efficiency, but also to ensure that hosts which move to different parts of the network are not permanently prevented from receiving frames [35]. That is, if the MAC-table update time is too large it is possible to lose the MAC address mapping from the switch MAC-tables so that only `null` is returned by the switchport resolution process (case 1 in Figure 4 and Table 1).

IEEE suggests an aging time of between 10 and $1,000,000$ seconds [35], with a suggested standard of 300 seconds. However, since unprocessed *wRecs* are buffered until the next MAC-table update, the logging process's memory utilisation ("working" memory - RAM) becomes an important and practical concern. With this in mind we choose a MAC-table update time of 5 seconds, which stays well within the suggested minimum aging time. This creates a worst case requirement of ≈ 300Mbytes of RAM on a full duplex 1Gbit/s link to buffer unprocessed *wRecs* , as we will see in our discussion of memory requirements in Section V.

Table 2: Summary - Skitter [38] mean RTT for all active servers on 3 arbitrary days

Day	Participating skitter nodes	Mean RTT for 99% of packets (ms)
02 Mar 2002	13	2050
02 May 2003	23	4709
02 Mar 2004	21	3941
RTT for 99% of packets over 3 days: 3566		

Purge interval length. The second timing parameter we consider is the active working records purge interval. This is the time after which the set of active working records is cleared and a corresponding set of connection records created and archived. As mentioned, the purge interval (*purgeInt*) is governed by an inherent time/space tradeoff. A larger *purgeInt* means a smaller proportion of frames become *conRecs* overall with each *conRec* representing a larger time interval. As can be seen in Figure 6, with a *purgeInt* of $1t$, *conRec* A 'represents' frames 1 and 2. If however the *purgeInt* were set to $2t$, then *conRec* A would have represented frames $1, 2, 3, 4$ and 5.

However, when replying positively to traceback requests (i.e., "yes I saw that frame") COTraSE will express the time that the given frame was processed in terms of purge intervals. Thus, whilst a larger purge interval decreases *conRec* storage requirements, it also decreases time precision of traceback replies. The purge interval must therefore be no greater than is acceptable for the purpose of traceback request processing. However, it must also be no smaller than the time we expect between processing a given frame and that frame reaching its intended destination as we explain below.

Traceback requests provide the time when a given frame to be traced was observed, either at the destination or at some suitable vantage point within the network (e.g., at an Intrusion Detection System). Based on this time we must search the connection record logs for those *conRecs* , if any, that represent the requested frame. As an example, consider Figure 6, where frame 1 is received and processed at the *conRec* log at time $0t + \delta$, where $0 \leq \delta < t$. Frame 1 reaches its destination at time $(0t+\delta)+\chi$ and so χ represents the 'delivery time'. If we ensure that $t \geq \chi$ then frame 1 will be delivered during the current or next purge interval (i.e., $0t \leftrightarrow 1t$ or $1t \leftrightarrow 2t$ respectively). If we assume that the delivery time is half the round trip time (RTT), we obtain that with a purge interval of $t \geq \frac{RTT}{2}$ the *conRec* for time $(0t + \delta) + \chi$ or the previous one 'represent' the tracked frame. The purge interval must be greater than half the expected round trip time (RTT) or equivalently the time we expect a frame to take in being delivered to its intended recipient.

To evaluate the impact of the chose *purgeInt* , we sampled data from the Skitter "Macroscopic Topology Project" [38] for 3 arbitrarily chosen days that are each one year apart.

Each data set shows the RTT for packets sent from a number of skitter nodes that were active on that day. For each node, we use the RTT distribution by continent, that is, the RTT from the given skitter node to servers around the globe (Africa, Asia, North America, South America, Middle East, Europe, Oceania).

We conservatively estimated that (at the $99th$ percentile) the round trip time is ≈ 3.5 seconds, as summarised in Table 2. For instance for data from 02 May 2003 and for each of the 23 available measurements we took the worst case (largest) round trip time at the $99th$ percentile and then calculated a mean value for all servers. Thus we feel that a choice of 10 seconds as the lowest bound on the purge interval is sufficient for $t \geq \chi$. Of course as 10s is the lower bound, it also produces the greatest number of connection records and so represents the worst case in terms of memory utility. We use this value in our implementation of the *conRec* log so that our discussion of memory requirements given later realistically portrays this worst case scenario.

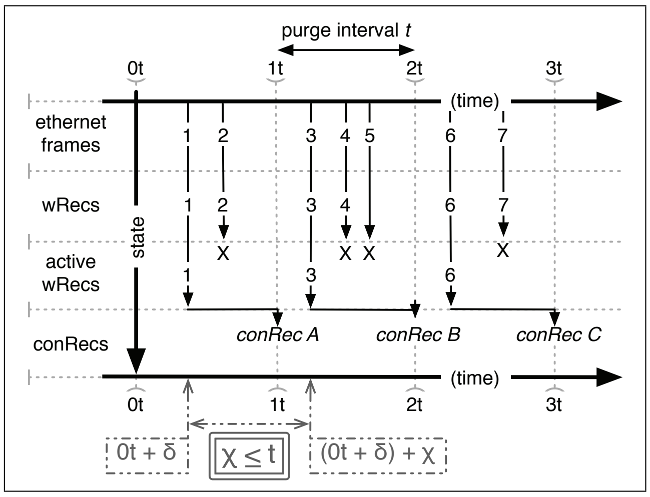


Figure. 6: The purge interval t must be greater than the time χ we expect a frame to take in reaching its destination

D. The Translation Record Logs

Traceback requests will specify a traced packet in the form that this was received by the recipient. As we have seen, where legitimate spoofing such as NAT is in place, the source IP address, source TCP port and destination TCP port are all subject to change at the gateway router. The translation records are a mapping between the connection identifiers before and after any of the address fields are re-written and CO-TraSE is the only L2 Traceback system we are aware of that explicitly addresses this issue. Each translation record contains *conId*_orig, *conId*_mod and a timestamp.

At the translation record (*transRec*) logs, the original connection identifier (*conId _orig*) is computed in the same way as the *conId* at the connection record logs. That is, a hashed digest over the concatenated source and destination addresses from the Transport, Network and Data-link layers (`TCP/UDP` ports, `IP` and `MAC` addresses, respectively). However, the modified connection identifier cannot include any data that is not received by a given frame's recipient. When servicing traceback requests, we need to match the data supplied by the trace initiator (i.e. the traced packet) with data we have stored in our logs. For frames that leave the local network, we expect the application, transport and network layer data to arrive unchanged (i.e. in the same form as when it left the local network). However we can make no assumptions about the physical (and by extension) data link layers employed to reach the intended destination (for instance ADSL or DOCSIS fibre links may be used). Thus, the source and destination MAC addresses are *not* included in the computation of *conId*_mod. The *transRec* logging algorithm is shown in Figure 7.

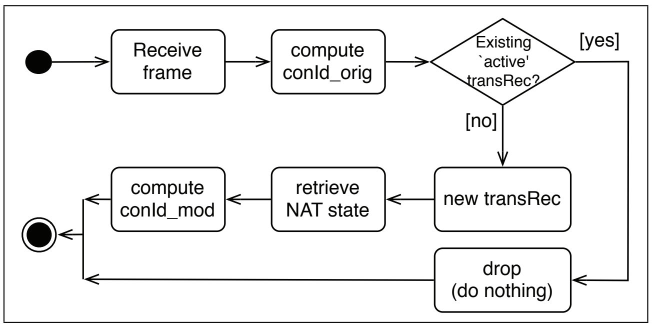


Figure. 7: The Translation Record logging process

The *transRec* log maintains an 'active connections' set in a similar manner to the *conRec* logs. The processing algorithms however are much simpler, in anticipation of the fact that the logging procedures at a leaf router may handle a greater volume of traffic. As can be seen in Figure 7 no attempt is made to perform switchport resolution as this is done at all *conRec* logs en route to the leaf router and doing it again provides no new information.

When a frame arrives at the router, the original connection identifier (*conId _orig*) is computed and a check made for an existing translation record in the 'active *transRecs* ' set with the same *conId _orig*. If there is a match then the given frame is no longer processed and 'dropped'. If however there is no existing active translation record, then a new *transRec* is created. The connection's state is retrieved (e.g., from the NAT processes or from the web proxy) and used to compute the modified connection identifier (*conId _mod*). A time-stamp is then added to this and the new *transRec* is placed in the leaf router 'active connections'. After the active connections purge interval has elapsed (as discussed above), the active connections set is cleared with the *transRecs* being archived to storage for subsequent traceback processing.

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 post-send legitimate spoofing at the leaf router.

V. Memory Requirements

Memory requirements are an important concern for logging based IP and L2 Traceback systems as they govern the 'traceback window', the time during which we can traceback a packet after it has been logged. That is, the length of time for which it is feasible to retain logged connection records, before the oldest *conRecs* are overwritten with new data. Below we give details of the various COTraSE records encountered so far as these are used in the discussion that follows (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)

Recall that the MAC-table update period directly impacts the 'working' memory requirements (RAM). All working records created from received frames are buffered to undergo switchport resolution only after the MAC-table update. Taking a duplex FastEthernet link at 1Gbit/s and with average frame size of 1000 bits, this results in $\approx 1,000,000$ frames per second in either direction. Assuming a hypothetical worst case scenario where all frames belong to different connections, this would equate to $\approx 10,000,000$ working records in 5 seconds. This scenario is of course highly unlikely and can only arise in a sustained DoS attack targeted specifically at COTraSE (further discussion of vulnerabilities follows separately). None the less, even in this extreme case a *conRec* log requires RAM of $\approx$ 300MBytes for 5 seconds worth of *wRecs* , which we do not consider excessive.
Storage requirements for archived connection records will ultimately depend on the number of conversations between peer machines and this will of course vary between networks. Recall that a single *conRec* is required for each ongoing connection for each purge interval. Values for the *conRec* storage requirements and, hence, overall memory utility are estimates. None the less, we have established some bounds using 'real world' network traffic from available WAN traces [28, 29, 30].
The number of connections C gives the lower bound for the number of *conRecs* cR we need to store for each purge interval of length t. 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 t, as explained earlier. A larger value of t will mean less *purgeInts* occur within a given traceback window and so less *conRecs* overall. As we have already seen, the lower bound for t, producing the greatest value for cR is 10 seconds. This value is used in our experiments so that our results represent the worst case, in so far as the purge interval can affect cR.

In total we processed over $300,000,000$ packets to derive an upper bound for the number of connections C. We cannot claim that our results represent a general case as of course C will vary greatly. Furthermore WAN traces are not generally representative of LAN traffic, which is the primary focus for COTraSE . However, 'real' LAN traffic is not easily available and one can appreciate the associated privacy and security concerns.
We first describe the WAN traces in more detail and explain how these were processed to produce a 'hex dump' for each minute of a given trace. This is followed by a description of our implementation of the *conRec* algorithm.

A. The WAN trace data

Table 3 summarises the trace data used in our analysis. Source 'OC12'[28] provides data from an OC12 link at the AMPATH Internet Exchange in Miami. We arbitrarily downloaded the data for 09 Jan 2007, with a total of 12 one hour traces taken at 3-hourly intervals from 0900 until 0000 of 10 Jan. Source 'OC48' [29] is an OC48 peering link for a large ISP at NASA Ames Internet Exchange. The data we use is for six 5 minute periods: two on each of 14 Aug 2002 (0900), 15 Jan 2003 (0959) and 24 Apr 2003 (0000).

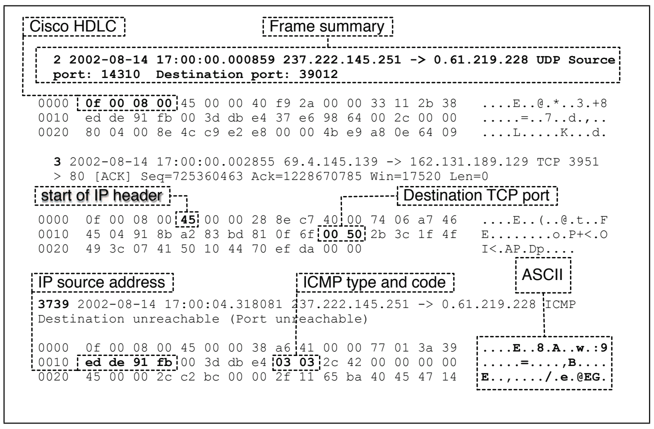
Table 3: Trace data summary

Source	Traces	Total Frames
OC12[28]	$12 * 60$mins	$136,503,068$
OC48[29]	$6 * 5$mins	$114,181,288$
WIDE[30]	$6 * 15$mins	$89,357,967$

All data was available in the form of `libpcap` [39] capture files and all traces are anonymised (e.g., IP addresses are changed) to preserve the privacy of network users. The OC48 and OC12 traces are captured separately in each direction of the monitored link; thus, the six traces for OC48 are in actuality either direction at 3 given times. Finally, source 'WIDE'[30] is a 100Mbit/s Ethernet transit link from the WIDE research network in Tokyo to its upstream carrying mainly trans-Pacific traffic. This source provides 15 minute traces and we used data for the same day (and times) as OC12. The WIDE capture files represent both directions of the monitored link.
Despite the WIDE data being duplex and the OC48 and OC12 simplex, we processed all trace files in the same way and treated the simplex traces as independent captures. Splitting the OC12 trace into two simplex traces would be difficult. However merging the simplex OC48 and WIDE traces was possible using the `mergecap` tool, available as part of the `wireshark` [40] network protocol analyser. In any case we decided that this was not necessary. Given the difference in both link speed and capture length for each given trace, it is not possible to directly compare the number of *conRecs* created even if all traces were either duplex or simplex. Further-

more, the *conRec* algorithm treats each direction of a given communication independently. A different connection identifier is derived for frames flowing from $A \rightarrow B$ than that derived for frames flowing $B \rightarrow A$. Thus, whether these are processed from a single file (in a duplex capture) or from two files (simplex) does not bear upon the total number of connection records created in the given time. As we will see our analysis compares the number of *conRecs* as a ratio of the total number of frames, to allow a direct comparison between the results for each source.

Each `.pcap` file was passed through the `tcpdump` utility to produce a text file complete with a 'hex dump' of each packet's data. An example is given in Figure 8 which shows data from an OC48 trace.


Cisco HDLC
Frame summary
start of IP header
Destination TCP port
IP source address
ICMP type and code
ASCII

```
  2 2002-08-14 17:00:00.000859 237.222.145.251 -> 0.61.219.228 UDP Source
  port: 14310  Destination port: 39012

0000  0f 00 08 00 45 00 00 40 f9 2a 00 00 33 11 2b 38   ....E..@.*..3.+8
0010  ed de 91 fb 00 3d db e4 37 e6 98 64 00 2c 00 00   .....=..7..d.,..
0020  80 04 00 8e 4c c9 e2 e8 00 00 4b e9 a8 0e 64 09   ....L.....K...d.

  3 2002-08-14 17:00:00.002855 69.4.145.139 -> 162.131.189.129 TCP 3951
  > 80 [ACK] Seq=725360463 Ack=1228670785 Win=17520 Len=0

0000  0f 00 08 00 45 00 00 28 8e c7 40 00 74 06 a7 46   ....E..(..@.t..F
0010  45 04 91 8b a2 83 bd 81 0f 6f 00 50 2b 3c 1f 4f   E........o.P+<.O
0020  49 3c 07 41 50 10 44 70 ef da 00 00               I<.AP.Dp....

3739 2002-08-14 17:00:04.318081 237.222.145.251 -> 0.61.219.228 ICMP
Destination unreachable (Port unreachable)

0000  0f 00 08 00 45 00 00 38 a6 41 00 00 77 01 3a 39   ....E..8.A..w.:9
0010  ed de 91 fb 00 3d db e4 03 03 2c 42 00 00 00 00   .....=....,B....
0020  45 00 00 2c c2 bc 00 00 2f 11 65 ba 40 45 47 14   E..,..../.e.@EG.
```

Figure. 8: Annotated Hex dump of an OC48 .pcap file, ready for processing by the conRec implementation

As can be seen the start of each frame is denoted by a frame summary before three lines show the base 16 representation of the packet's content, with the ASCII equivalent to the right (non printing ASCII characters are shown as '`.`'). Each two digit hex code represents a single byte of raw network data. For the OC48 traces the Cisco HDLC data link layer was employed. As can be seen in Figure 8, the code 0800 signals that an IP frame follows. Code 45 follows to signal the start of the given IP header. This represents the first two fields of the IP header, showing the IP version as 4 and the IP header length as 5. As these fields each occupy a nibble, they are shown together as a single byte in the hex dump. Furthermore, this is interpreted as the character `E` in the ASCII representation of the data as can be seen.

Each trace file was processed by tcpdump in the same way to produce an equivalent text file complete with the hex representation. Our Java implementation of the *conRec* logging algorithm takes these files as input and parses the hex byte codes to reproduce the necessary fields for working and connection records.

B. Results

The primary purpose of this investigation is to empirically deduce a bound on the number of connections C against real network data. Recall that the number of connections C gives the lower bound for the number of *conRecs* cR we need to store for each purge interval of length t.

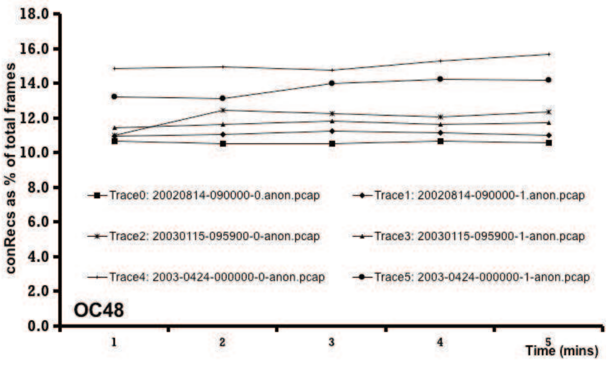
For each source our code creates a *conRec* file for every 10 second purge interval. A separate 'debug' file was also generated for each trace file. The *conRec* and debug files were then used to collate the total number of connection records calculated for each minute (i.e. from each of the six purge intervals occurring therein). Table 4 summarizes some of the characteristics of the network data, as revealed by our processing.

'Bytes per packet' was taken from the metadata associated with each capture file as indexed at DatCat [41]. This is significant as in our subsequent calculations we compute the maximum number of frames based on an average frame size of 1000 bits. As can be seen in Table 4 this is a conservative estimate, since the average packet size from the WAN data being more than five times that at 673 bytes per packet. The 'packets per second' column was populated by taking the total number of working records created by the data from each source, and dividing by the total capture time. For example for OC48 we processed $136,803,068$ packets occurring over 15 minutes (processed as 30 minutes of simplex captures), equating to $128,868$ packets per second. The '% unparseable' represents those frames that our parser classes skipped; recall that our code only processed all `TCP`, `UDP` and `ICMP` data encountered. As can be seen, IPv6, ARP and incomplete frames (e.g. due to an error when capturing) accounted for a very small fraction of the total number of frames.

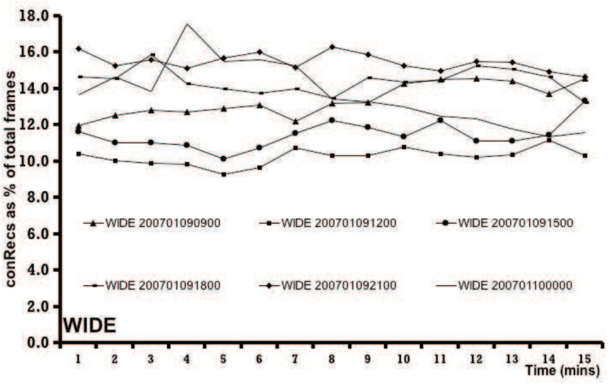
Table 4: Characteristics of trace data as revealed by our implementation

Source	bytes per packet	packets per second	% unparseable !(TCP, UDP, ICMP)
OC48	563	$128,868$	0.45
OC12	779	$6,320$	2.76
WIDE	676	$16,548$	1.07

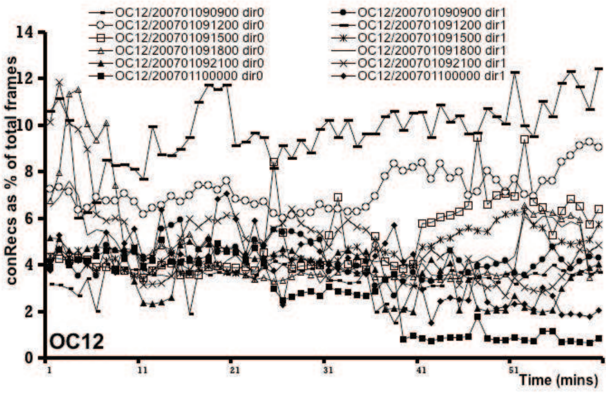
We plotted the number of *conRecs* as a percentage of the total number of frames for each minute of each trace, as shown in Figure 9, depicting result for OC48, WIDE and OC12, respectively. Obviously a lower $\frac{conRecs}{frames}$ ratio signals a greater reduction in memory requirements compared to logging all frames. We see a similarity in the plots and especially so between Figure 9(a) and 9(b), where the number of *conRecs* is roughly between 10 and 15 percent of the total number of frames. The OC12 trace in Figure 9(c) shows even larger improvement, with many data points as low as 4 percent. We conclude from these experiment that it is not unreasonable to expect one order of magnitude memory reduction using



(a) CAIDA OC48 [29]



(b) WIDE [30]



(c) CAIDA OC12 [28]

Figure. 9: *conRecs* as a percentage of frames for each minute of the given trace

COTraSE compared to logging all frames.

Note that the results in Figure 9 are for a purge interval of 10 seconds. By enlarging the purge interval, storage of *conRecs* can be further reduced, with a theoretical optimum when the whole trace is treated as one single purge interval. The traces are too large to conduct this analysis, and as we have argued in Section IV-C, larger purge intervals reduce the accuracy of the traceback since the time the frame was sent is reported in terms of the purge interval.

We use $\frac{conRecs}{frames}$ to approximate the worst case number of connections for a fully loaded network. From Figure 9 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, which as we have seen is a conservatively low estimate. In Table 5 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). As we have already stated it is not possible to derive a general case from this data. The number of connections will vary greatly though it is encouraging that the values we obtained from this 'real world' data are consistently low, with the number of *conRecs* an order of magnitude less than the total number of frames.

Table 5: 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	$1,933$

VI. Discussion and Deployment Issues

As mentioned earlier, there are similarities between the 'connection-oriented' logging approach taken by COTraSE and Cisco's 'NetFlow' [37, 42]. NetFlow is deployed in Cisco routers and some high end 'multilayer' switches. In this section we briefly consider NetFlow and the associated IPFIX standard [43, 44] and reveal elements of its operational model that make it unsuitable for L2 Traceback. Finally we discuss some of the deployment issues for COTraSE before concluding.

A. NetFlow and IPFIX

NetFlow is a general purpose traffic reporting system that may be used to determine the composition of traffic on a network [45]. In its simplest form, a 'flow' consists of source/destination IP addresses, TCP ports and the IP protocol field. IPFIX defines a standard format for exported data and the protocol by which these are delivered for analysis.

The main difference between COTraSE and NetFlow/IPFIX lies in the expiration of flows. In NetFlow, flows are expired if: the end of a flow can be detected (i.e. TCP 'FIN' or 'RST'), when the flow has been inactive for a configurable timeout, or due to resource constraints (e.g. memory exhaustion). Longer lasting flows are also expired periodically (suggested 30 minutes) to avoid 'staleness'. In COTraSE however, all 'current' flows (or 'connections' as per our terminology) are expired at the end of the current 'active connections purge interval' as we have seen. We note that a similar approach is suggested in [45], where the authors suggest a much larger 'time-out' than we do (their focus was on improving NetFlow rather than L2 Traceback).

There are some limitations of NetFlow, limiting its direct applicability to traceback.

Flows are reported after the event: Flows are reported when they expire, or after 30 minutes (configurable). Thus, the utility of this data by IDS or security monitoring systems in general is debatable.

Memory constraints often force a sampling approach: The metering process maintains state for all active flows; with a large number of (longer lasting) flows, memory utilization can quickly increase beyond available levels. Sampling of traffic is suggested as a potential solution, and this is unacceptable for Traceback systems in general.

The exporting process is expensive and vulnerable: Flows are reported by the metering processes to the collection host as they expire. Given the high frequency of this operation, strong authentication and encryption of transferred data become expensive. A 'dedicated' link is suggested as a potential solution,

Flow expiration of non-TCP flows is inaccurate: This is well known, and reported in [45]. Periodically expiring all active flows, as in COTraSE, improves accounting of non-TCP flows and removes the requirement to examine TCP flags when these are present.

We also note that NetFlow/IPFIX do not make provisions for legitimate spoofing and do not specify a process for switchport resolution.

B. *COTraSE deployment issues*

Performance: Maintaining the connection identifier 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 [31] 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. This point will be further discussed in the next chapter.

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 such functionality is currently not made available by ethernet switch manufacturers.

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. As was suggested in Chapter 4, the MAC-table update time can be randomized (e.g., between 3 and 5 seconds). Furthermore, COTraSE uses the MAC-tables from both switches to perform switchport resolution making conditions even more adverse for an attacker.

Finally, an obvious attack on the COTraSE system is for an adversary 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*) which can at least aid in the identification of the instigating origin.

VII. Conclusion

We introduced COTraSE, a 'connection oriented' logging based Layer 2 traceback system for Switched Ethernet. COTraSE allows traceback of ethernet frames that are wholly local but also for IP packets addressed to external recipients, regardless of spoofed MAC or IP addresses and any address translation mechanisms (e.g., NAT). Rather than explicitly logging all traffic COTraSE attributes frames to ongoing connections and logs 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'. COTraSE offered improvement over our earlier switch-SPIE system in a number of areas, most importantly increased reliability and decreased memory requirements. 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 results show a saving of about an order of magnitude in frame storage requirements compared to exhaustive logging.

References

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

[2] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *ACM SIGCOMM Computer Communication Review*, 30(4):271–282, 2000.

[3] A. C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F Tchakountio, B. Schwartz, S.T. Kent, and W.T. Strayer. Single packet ip traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, 2002. preliminary version presented at ACM SIGCOMM 2001.

[4] Hal Burch and Bill Cheswick. Tracing anonymous packets to their approximate source. In *14th USENIX Conference on System Adminimstration (LISA)*, 2000.

[5] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for ip traceback. *IEEE/ACM Transactions on Networking*, 9(3):226–237, 2001.

[6] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In *IEEE Workshop on Information Assurance and Security*, 2001.

[7] S. Bellovin, M. Leech, and T. Taylor. Icmp traceback messages - ietf internet draft. `http://www3.tools.ietf.org/html/draft-ietf-itrace-00`, 2000.

[8] A. Koyfman, T. Doeppner, and P. Klein. Using router stamping to identify the source of ip packets. In *ACM Conference on Computer and Communications Security CCS*, 2000.

[9] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *IEEE INFOCOM*, pages 878–886, 2001. Vol. 2.

[10] Kihong Park and Heejo Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *12th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 338–347, 2001. volume 1.

[11] Michael T. Goodrich. Efficient packet marking for large-scale ip traceback. In *9th ACM conference on Computer and Communications security*, pages 117–126, 2002.

[12] A. Mankin, D. Massey, W. Chien-Lung, S.F. Wu, and L. Zhang. On design and evaluation of intention driven icmp traceback. In *10th International Conference on Computer Communications and Networks*, pages 159–165, 2001.

[13] Vadim Kuznetsov, Helena Sandström, and Andrei Simkin. An evaluation of different ip traceback approaches. In *4th International Conference on Information and Communications Security*, pages 37–48, 2002.

[14] E. Jones, O. Le Moigne, and J. M. Robert. Ip traceback solutions based on time to live covert channel). In *12th IEEE International Conference on Networks ICON*, pages 451–457, 2004. Vol. 2.

[15] A. Belenky and N. Ansari. Tracing multiple attackers with deterministic packet marking (dpm). In *IEEE PACRIM Conference on Communications, Computers and Signal Processing*, pages 49–52, 2003. Vol. 1.

[16] T. Baba and S. Matsuda. Tracing network attacks to their sources. *IEEE Internet Computing*, 6(2):20–26, Mar/Apr 2002.

[17] H. C. J. Lee, Ma Miao, V. L.L. Thing, and Yi Xu. On the issues of ip traceback for ipv6 and mobile ipv6. In *IEEE International Symposium on Computers and Communication*, pages 582–587, 2003.

[18] K. Shanmugasundaram, H. Brönnimann, and N. Memon. Payload attribution via hierarchical bloom filters. In *11th ACM Conference on Computer and Communications Security*, pages 31–41, 2004.

[19] T-H Lee, W-K Wu, and T-Y W. Huang. Scalable packet digesting schemes for ip traceback. In *IEEE International Conference on Communications*, pages 1008–1013 Vol.2, 2004.

[20] L. Zhang and Y. Guan. Topo: A topology-aware single packet attack traceback scheme. In *International Conference on Security and Privacy in Communication Networks*, 2006.

[21] C. Gong, L. Trinh, T. Korkmaz, and K. Sarac. Single packet ip traceback in as level partial deployment scenario. In *IEEE Global Telecommunications Conference*, 2005.

[22] V. L. L. Thing, M. Sloman, and N. Dulay. Non intrusive ip traceback for ddos attacks. In *ACM Symposium on Information, Computer and Communicat ions Security ASIACCS*, pages 371–373, 2007.

[23] J. Li, M. Sung, J. Xu, and L. Li. Large-scale ip traceback in high-speed internet: Practical techniques and theoretical foundation. In *IEEE Symposium on Security and Privacy*, pages 115–129, 2004.

[24] R. Chen, J-M.Park, and R. Marchany. Track: A novel approach for defending against distributed denial-of-service attacks. Technical report, Virginia Polytechnic Institute and State University, 2005. TR-ECE-05-02 Dept. of Electrical and Computer Engineerig.

[25] H. Hazeyama, M. Oe, and Y. Kadobayashi. A layer-2 extension to hash based ip traceback. *IEICE Transactions on Information and Systems*, E86(11):2325, 2003.

[26] M. Snow and J. Park. Link-layer traceback in switched ethernet networks. In *IEEE LANMAN*, pages 182–187, 2007.

[27] Marios S. Andreou and Aad van Moorsel. Logging based ip traceback in switched ethernets. In *EUROSEC '08: Proceedings of the 1st European workshop on system security*, pages 1–7. ACM, 2008.

[28] C. Shannon, E. Aben, K.C. Claffy, and D. Anderson. The caida anonymized 2007 internet traces, 2007. http://www.caida.org/data/passive/passive_2007_dataset.xml.

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

[30] WIDE Project. Packet traces from wide backbone, mawi working group traffic archive, 2008. http://mawi.wide.ad.jp/mawi/.

[31] Council European Parliament. On the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks. *Official Journal of the European Union*, L 105:54–63, April 2006. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006L0024:EN:NOT.

[32] M. Andreou and Aad van Moorsel. Cotrase: Connection oriented traceback in switched ethernet. In *The Fourth International Conference on Information Assurance and Security*, 2008.

[33] Ahmad Almulhem and Issa Traore. A survery of connection-chains detection techniques. In *IEEE PacRim Conference on Communications, Computers and Signal Processing*, pages 219–222, 2007.

[34] Rich Seifert. *The Switch Book: The Complete Guide to LAN Switching Technology*. John Wiley & Sons, Inc., 2000.

[35] IEEE. Ieee standard for local and metropolitan area networks: Media access control (mac) bridges. Technical Report 802.1D - 2004, IEEE, Feb 2004.

[36] BBN Technologies. Source path isolation engine, 2004. http://www.ir.bbn.com/projects/SPIE/spiehome.html.

[37] B. Claise. Cisco systems netflow services export version 9 - ietf rfc 3954, informational. http://www.ietf.org/rfc/rfc3954.txt?number=3954, 2004.

[38] CAIDA Macroscopic Topology Project Team. CAIDA skitter Topology Traces (collection), 2007. http://www.caida.org/tools/measurement/skitter/.

[39] Van Jacobson, Craig Leres, and Steven McCanne. tcpdump, 2009. http://www.tcpdump.org/.

[40] Gerald Combs et al. Wireshark network protocol analyser, 2009. http://www.wireshark.org/.

[41] Cooperative Association for Internet Data Analysis CAIDA. Internet measurement data catalog, 2009. http://www.datcat.org.

[42] CISCO Systems. Cisco ios netflow white papers. http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html, 2008.

[43] J. Quittek, T. Zseby, F. Fokus, B. Claise, and S. Zander. Requirements for ip flow information export (ipfix) - ietf rfc 3917, informational. http://www.ietf.org/rfc/rfc3917.txt?number=3917, 2004.

[44] B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information - ietf rfc 5101, standards track. http://www.ietf.org/rfc/rfc5101.txt?number=5101, 2008.

[45] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *ACM Special Interest Group on Data Communication SIGCOMM '04*, pages 416–428, 2004.

Author Biographies

Marios Andreou was born in London on 1 June 1981. He received a BSc. (hons) in Software Engineering from the School of Computing Science at Newcastle University in 2004. In 2004 he conducted a placement at Hewlett-Packard Laboratories in Bristol, UK, where he explored virtualisation technologies for utility computing. He is currently a teaching assistant and PhD candidate at Newcastle University working in the area of network security and message traceback systems.

Aad van Moorsel is a Reader (Associate Professor) in Distributed Systems at the University of Newcastle. Prior to joining Newcastle he worked in industry for almost a decade, first as a researcher at Bell Labs/Lucent Technologies in Murray Hill and then as a research manager at Hewlett-Packard Labs in Palo Alto. There he was responsible for HP's research in web and grid services, and worked on the software strategy of the company. Aad got his Ph.D. in Computer Science from Universiteit Twente in The Netherlands (1993) and has a Master's degree in Mathematics from Universiteit Leiden, also in The Netherlands. After finishing his PhD he was a postdoc at the University of Illinois at Urbana-Champaign for two years. He has worked in a variety of areas, from performance modeling to systems management, web services and grid computing. He has published more than 70 peer-reviewed conference and journal papers, received several best-paper awards, owns three US patents and gained various company awards. His research interest is in creating the 'intelligent enterprise' through applying quantitative methods and techniques (including monitoring) to information systems and technology decision making.