

Link-Layer Traceback in Ethernet Networks

Michael Snow and Jung-Min Park
Virginia Polytechnic Institute and State University
{sno, jungmin}@vt.edu

Abstract – *The design of the most commonly-used Internet and Local Area Network protocols provide no way of verifying the sender of a packet is who it claims to be. A malicious host can easily launch an attack while pretending to be another host to avoid being discovered. To determine the identity of an attacker, an administrator can use traceback, a technique that determines the path of attack packets from the victim to the coordinator. Most traceback research has focused on IP and Stepping-Stone techniques and little has been conducted on the problem of Data-Link Layer Traceback (DLT), the process of tracing frames from the network edge to the attack source. We propose a scheme called Tagged-fRAme tracebaCK (TRACK) that provides a secure, reliable DLT technique for Ethernet networks. TRACK defines processes for Ethernet switches and a centralized storage and lookup host. Simulation results indicate that TRACK provides accurate DLT operation while causing minimal impact on network and application performance.*

I. INTRODUCTION

Internet-based attacks are increasingly common in today's networks. Tools are readily available that allow even inexperienced users to launch sophisticated attacks. In addition, the most popular and widely-used networking protocols allow a user to disguise his or her true identity and location.

The only reliable way to determine the machines involved in the attack is to trace the entire path of the attack, usually a three-stage process. The first stage, IP traceback, is to trace the path from the victim machine to the edge of the network from which the attack packet originated. The second stage, DLT, is from the router at the edge of that network to the machine that actually sent the packet, the zombie. The third step, Stepping-Stone Traceback, is tracing the flow of control packets to the zombie from the attack coordinator.

IP traceback, by design, will only return the location of the edge router for the subnet in which the zombie machine is located. This is because the routing of packets within the Internet is determined by Layer 3 (L3) address information. Within the subnet, addressing is most likely based on Layer 2 (L2) information. Once the packet exits the L2 network, the L2 source and destination addresses will likely change at each L3 hop. Additionally, the packet may be transmitted over many different types of L2 links which have completely different frame and address formats. For a traceback scheme to effectively locate a host in an L2 network, it would have to be able to map a received IP packet to a host based on L2 information which had been previously collected. However, since source address information cannot be trusted, such a scheme must be able to determine where the device is physically connected to the network.

While IP Traceback and Stepping-Stone Traceback are well-researched areas, very little work has been done in the area of Data-Link Layer Traceback (DLT). In 2003, Hazeyama et al. published the xDGA scheme to support link layer traceback [1]. This scheme focuses on minimizing storage requirements and strict adherence to established networking standards. Though this allows for easy implementation in almost all existing Ethernet networks, it comes at the expense of high computational overhead and possibly inaccurate results. The scheme also assumes that a particular IP Traceback method, SPIE [2], is supported at the edge router.

We propose a DLT scheme called *Tagged fRAme tracebaCK*, TRACK, designed with the goals of increasing result accuracy, decreasing edge router loading, and removing the dependence on the SPIE scheme. TRACK defines an extension to the Ethernet frame format to tag frames as soon as they enter the network. Those tags are collected by a dedicated process at the network edge.

This paper is organized as follows: Background information, including applicable networking standards and related research, is discussed in Section II. In Section III, the network and attack models and assumptions are discussed. In Section IV, we discuss the architecture and operation of the TRACK protocol and associated devices. Section V describes the simulation and results. Future work is discussed in Section VI and Section VII concludes this paper.

II. RELATED WORK

A. IP Traceback

IP Traceback refers to the process of tracing a single packet or a flow of packets from the victim back through the network to the edge of the network from which the packet or flow originated. This is a very well researched area and many schemes have been proposed [2][3][4][5][6][7][8].

One type of scheme involves packet tagging. Such schemes have been proposed by Savage et al. and Song et al. [3][4]. In these schemes, packets are tagged as they pass through routers in the network. A router may tag every frame or tag frames using a probabilistic function. The tag may contain complete or partial path information. Another host is responsible for collecting and analyzing these tags to build an attack path for a given packet or flow. Such schemes may provide authentication for packet tags to verify the identity of the tagging router [4].

In the second type of scheme, each router in the network collects data on the packets that it forwards. That data is aggregated by another host or set of hosts that process the data to form the attack path for a packet or flow. The *Source Path Isolation Engine* (SPIE) proposed by Snoeren et al. [2] is one

such scheme.

B. xDGA: A Layer-2 Extension to SPIE

This scheme, proposed by Hazeyama, et al., extends the SPIE architecture to enable DLT operation. The scheme allows the packet ingress point to be narrowed down to a switch and port number on wired, switched Ethernet networks [1]. This information is determined through SNMP queries that retrieve the forwarding databases (FDB) of the network switches. This data is used to map the MAC address of an incoming packet from the subnet to a switch and port number and, optionally, a VLAN identifier.

When a traceback request implicates the subnet on which xDGA is running, a packet digest is generated from the suspect packet. This digest maps to an entry in the SPIE L3 digest table. That entry is then mapped to an entry in an L2 digest table which will indicate on which switch and port number the packet entered the subnet. This process involves several rounds of hashing and inspecting data and is $O(n^2)$.

Additionally, the authors point out a weakness in the protocol design of fixed-interval sampling of switch data. Ideally, any time the FDB data is updated, the xDGA would be made aware of this change. However, this may contribute to significantly increased network overhead. With periodic sampling, however, changes in FDB data may not be picked up by the xDGA quickly enough or at all. An intelligent attacker can use this property of the proposed xDGA design to launch an attack without being detected.

III. TAGGED-FRAME TRACEBACK (TRACK) ARCHITECTURE

Tagged-fRAME tracebaCK (TRACK) is a link-layer traceback protocol for wired Ethernet networks. To provide highly accurate results, TRACK uses frame tagging to map a suspected packet to the switch and port number where it entered the layer-2 subnet, regardless of the original frame's source MAC address. Frame tagging operations are based on the IEEE 802.1Q [9] and 802.1ac [10] standards, using the VLAN tagging model and configuration information defined by those protocols.

TRACK defines a TraceBack Tag (TBT) that is added to an L2 frame as it passes through a TRACK-enabled device. To facilitate traceback, two processing entities are defined. Within each switch is the TRACK Frame Tagger (TFT) that is responsible for applying tags to and removing tags from frames that pass through it. At the network edge nearest the router is the TRACK Analysis and Collection Host (TACH). This host collects frame tags before the frame is passed to the edge router.

A. TraceBack Tag (TBT) Format.

The traceback tag is added to a frame by the TFT and collected by the TACH. When a TBT is added to a frame, the contents of the frame are shifted down from the insertion point immediately after the 802.1Q tag. The rest of the original frame follows. The original LENGTH/TYPE field is set to indicate that an 802.1Q tag and TBT are present. The allowed MTU of any ports carrying TBT-tagged frames must be increased from 1522B to 1535B to support the addition of the new tag. Finally, the M_UNITDATA structure defined for

frame representation within bridges by the IEEE 802.1D standard should be modified to include a field for the TBT. This will be referred to as *traceback_tag*, in keeping with syntax used in the standard.

The TBT contains the fields used to support traceback operations. These 13 octets (Bytes) are broken into four fields. The first four bits represent the FLAGS field which consists of bits that are used to control traceback operation and data storage, as described in the following paragraph. The next 12 bits, labeled PORT_ID, contain a numeric value representing the port number from which the frame originated. This allows up to 4096 ports per switch, the same length used by Hazeyama et al. [1]. The next 8 bits, labeled SWITCH_ID, is used to store the ID of the switch the host is connected to. This field allows a network of up to 256 switches. Together these two fields uniquely identify the location of a host on the wired network. The remaining 80 bits represent the AUTH field and are used to provide authentication for the TBT.

The FLAGS field contains three control flags and an additional bit for future use. The (MSb), bit 7, is the Previously Tagged flag (PREV). Bit 6 is the Core Switch (CORE) flag. The FAIL bit, bit 5, is used to indicate an authentication failure and is reserved for use at the TACH. The remaining bit is unused and must be set to 0. The process of adding and removing tags and setting the FLAGS bits is described in Section 4.

B. TRACK Frame Tagger (TFT).

Each TRACK-enabled switch in the network contains a TRACK Frame Tagger (TFT). The TFT is responsible for adding a tag to or removing a tag from a given frame, based on the port on which the frame was received, the port on which the frame is scheduled for transmission, and the switch configuration. TFT processing should be executed after the transmission port(s) has been determined but before the frame is actually queued at the port(s) for transmission. Based on the 802.1D [11] and 802.1q [9] standards, selection of transmission port(s) occurs at the Frame Filtering stage of the forwarding process. IEEE 802.1q adds another step, Egress, where frames are actually queued at the transmission ports and before which TFT processing must occur.

C. TRACK Analysis and Collection Host (TACH).

At the edge of the network, the frame tags are collected by the TRACK Analysis and Collection Host (TACH). The frames are stored in a table mapping the frame contents to a switch and port number for later use if the frame is implicated by an IP Traceback request. The L2 table must provide a quick, efficient method of storing entries and retrieving them later. To support both, the L2 table is actually composed of the L2 digest table and host table, with links defined between the two.

The L2 digest table stores frame digests based on the first 32 Bytes of layer-3 data within the frame. This includes the fields of the IP header, masked as described by Snoeren et al. [2], and the first 12 Bytes of payload. The 32-Byte length, according to analysis performed by the same authors, is sufficient to provide a packet collision rate of less than 1 in 10,000

packets in a LAN [2]. According to their analysis, significantly better collision reduction was not possible. Each entry in this table also contains a timestamp indicating when the tagged frame was, a field for the TBT flags, and a link back to the record indicating which host sent the frame. Assuming a link rate of 100Mbps into the TACH entity, this table would require at most about 405MB of storage space for every minute of digests stored. Assuming an average frame size of 1000 bits, the storage space required is only about 150MB. The TACH also maintains the time at which the L2 table was last paged out to long-term storage, if this is made available.

The host table maintains an entry for each known switch/port ID combination on the network. Given the limitations imposed by the TBT format, there will be at most 2^{20} (1M) different entries in this table which should be sufficient to represent any LAN. With a host table entry size of 2B, this would require about 2MB of space to represent. A copy of this table should be stored along with the L2 table when it is paged out. However, because of the relatively small size of this table and the slightly higher cost of addition, described below, only the L2 table links need to be cleared from the in-memory copy of this table.

To support fast insertions, the host table should be sorted and the L2 digest table left unsorted. Additions to the host table, which should be relatively infrequent, will take longer than a simple unsorted addition, but the time to look up an entry, which is a much more common occurrence, will be greatly improved. L2 entry table additions can simply be added to the end of the table.

IV. TRACK OPERATION

This section discusses the operation of the TFT and TACH under normal conditions and when a traceback request is received. The switch containing the TACH module is referred to as the “root” switch since it is located nearest the edge router, though this may not be the root of the active spanning tree.

A. TFT Operation

TFT processing follows the flowchart shown in Figure 1. Ports on which frames are transmitted untagged are referred to as “access ports” and ports on which frames are transmitted tagged as “trunk ports.” To determine whether a port is an “access” or “trunk” port, the algorithm checks the VLAN port map entries.

The normal path of execution at a leaf switch – one connected to end hosts – will be that an untagged frame is received on an access port and scheduled for transmission on one or more trunk ports. In that case, a TBT will be added to the frame with CORE and PREV cleared. If the frame is scheduled for transmission only on other access ports, no TBT will be calculated or added to the frame. If a frame arrives on an access port already tagged (perhaps due to malicious behavior of other nodes) and is scheduled for transmission on at least one trunk port, the previous tag is removed. A new tag is calculated with PREV set. If

the frame is scheduled for transmission only on access ports, no tag is calculated.

The normal path of execution at a core switch – one connected between switches or at the root of the spanning tree closest to the edge router – involves receipt of tagged frames on a trunk port. This operation is dependant on the implementation and the placement of the TACH and setting of the configuration variable TACH_LOC. If TACH_LOC is set to a port number, then an unaltered copy of the frame, including the TBT, will be sent to this port. A copy of the frame with the TBT removed will also be transmitted on the scheduled port(s) determined by the forwarding process. If TACH_LOC is set to INLINE, then the frame will be transmitted with the TBT intact on any trunk ports scheduled for transmission (the TACH is between this switch and the router). The tag will be stripped before the tag is transmitted on any access ports. If TACH_LOC is set to INTERNAL, then the frame will be transmitted untagged on any ports scheduled for transmission. An unaltered copy of the frame will be directed to the internal TACH processor. If a frame is received at a core switch on a trunk port untagged and is scheduled for transmission on at least one trunk port, a TBT will be added to the frame with CORE set. If the frame is scheduled for transmission on access ports only, no tag will be calculated. In all cases, no tagged frame will be transmitted on an access port.

B. TACH Tag Collection

Generally the TACH will operate collecting frame tags as the frames pass through to the edge router. Once the frame is read in, the TACH creates an L2 table entry object with an appropriate timestamp and copies the FLAGS field from the TBT to the new entry. The TACH then calculates the AUTH code for the frame based on the method discussed in Section IV.D. This value is compared to the value in the TBT. If the two match, the FAIL bit in the FLAGS field is cleared. If not, the bit is set. The packet digest is calculated and stored in the new L2 table entry. The entry is then added to the L2 table.

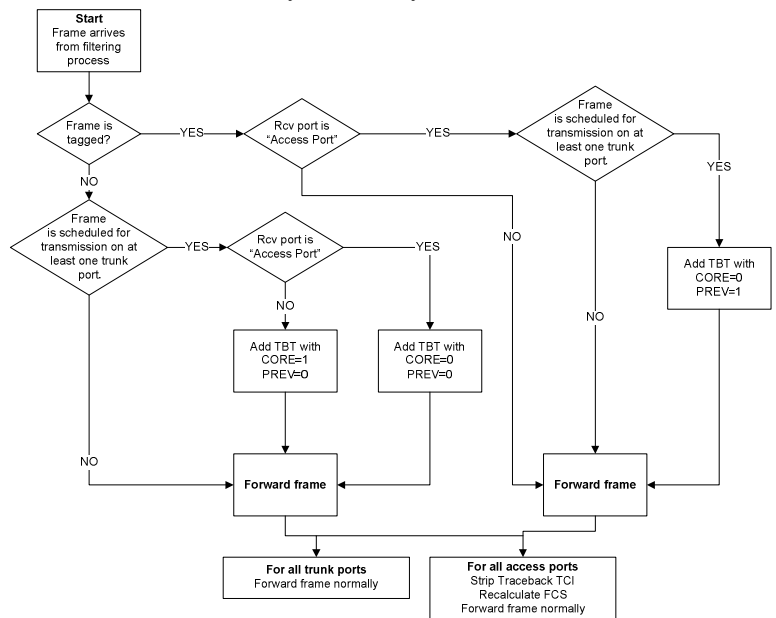


Figure 1 – TFT process flowchart

The host table is searched to find the entry corresponding to the Switch ID and Port Number from the TBT. If an entry is found, the link is set from the new L2 table entry to the existing host entry and a link to the L2 table entry is added to the host entry. If no host entry exists, one is inserted in the table, preserving the condition that the table remain sorted, and a link is added from that entry to the L2 entry.

C. TACH Traceback Request Processing

A TRACK traceback request shall consist of at least the 32 Bytes of the suspect packet at the time at which it was received at the victim. The packet digest for the packet received will be calculated as described in the following section. The TACH will search the L2 table corresponding to the request timestamp to find the corresponding entry. If one is found, the corresponding host record will be retrieved. Based on the status of the FLAGS bits, the two together will implicate either a specific host or a nearby switch. If FAIL is set, the tag information may or may not be able to be trusted. If the CORE bit is set, the entry most likely implicates a nearby switch, rather than the switch to which the host is directly connected. If PREV is set, the end host most likely attempted to thwart the traceback system by applying its own false traceback tag. Decisions for use and display of this information are left to the implementer. The Switch ID and Port Number from the host table are reported to the user as the host suspected of sending the packet. If an L2 digest table entry is not found for the packet, the previous (paged) table may be checked for the entry using the same process. If an entry is not found in that table, it is likely that the packet did not originate from this network.

In the event of L2 entries with duplicate digest values, the timestamps will be checked. Any entries with timestamps within a preset range of the given timestamp will be returned. The likelihood of duplicated digest values with similar timestamps, however, is assumed to be low. In the event that the TACH returns records from multiple sources with similar timestamps, it is likely that multiple attacking hosts sent copies of the same frame toward the destination.

D. Protocol Security

To guarantee authenticity of the TBT, TRACK uses a keyed HMAC to authenticate the frames to which a TBT is applied. To support this, TRACK requires that a key or keys be established between all entities supporting the TRACK protocol. This key may be shared between all entities or separate keys may be established on a pairwise basis. Since the TACH is responsible for verifying authentication information, it should also be responsible for establishing and maintaining keys. It is assumed that switches participating in TRACK can be trusted.

The AUTH field is calculated using a Hashed Message Authentication Code (HMAC), as defined in RFC 2401 [12]. It is assumed that SHA-1 will be used as the underlying hash function. The output of the function is truncated to 80 bits to reduce network overhead. Any hash function may be used as long as the output is truncated to no less than half of the hash result [12]. If a network-wide key is used, the TFT will use that key to generate the HMAC which will then be truncated

and stored in the AUTH field of the TBT. The TACH will use the same key and hash function to compute the authentication information to compare to the TBT value. Otherwise, the TFT will use its assigned key to compute the AUTH field data as described previously. The TACH will use the key corresponding to that switch to calculate the authentication. In either case, the data input to the function is at least the first 32 Bytes of the frame.

V. SIMULATION FRAMEWORK

To test the effects of TFT processing on network performance, a version of the TFT was implemented in the OpNET [13] switch model. The simulated network consisted of 15 clients connected through a network of 5 switches to a single server, as shown in Figure 2. The TACH entity was simulated. However, the impact of the TACH on network operation should be minimal. The primary concerns are the additional delay and frame size created by the TFT processing.

All communications in the network are unicast between the clients and the server. To simulate the traffic load through the root switch, the server is located behind the root switch. All links are 100Mbps. Redundant links are handled by the MSTP [9] running normally between the switches. Each simulation considers 3 hours of network operation.

Four different scenarios are considered. The baseline case has TRACK disabled. Standard Ethernet frames are used and transmitted with VLAN tags per the network configuration (VLAN tags on trunk links). The remaining three test network performance with TRACK enabled. They use the larger frame size and test three different processing delay values. The “fast” TFT implementation is set to require 9.25µs to tag a frame. This represents the processing delay of a dedicated hardware implementation [14], though faster ones are available [15]. The “medium”-speed scenario uses a processing time of 34.83µs. This represents the amount of time required for a reasonably fast software hash function implementation [4]. The longest delay specified was 143µs for the “slow” mode, which is scaled from the previous value, assuming a reasonably fast embedded processor. In Fig. 3, the aforemen-

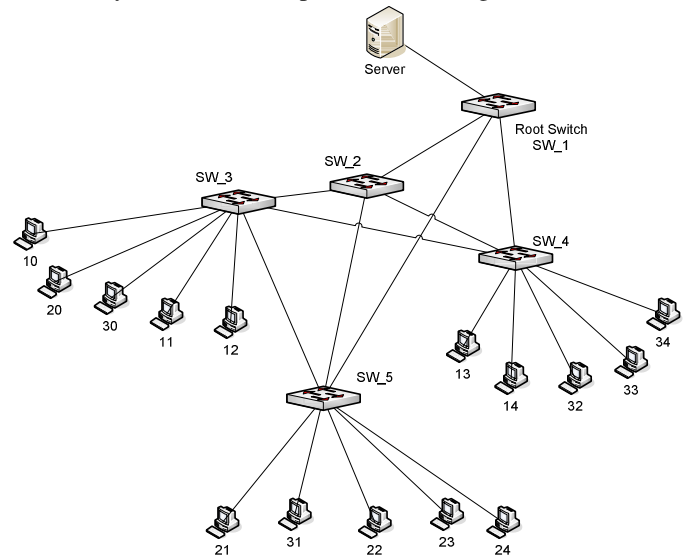


Figure 2 – Simulation network

tioned scenarios are represented as TRACK(9.25), TRACK(34.83), and TRACK(143), respectively.

Three different traffic class configurations are simulated. In the first class, all nodes simulate a user browsing a series of web pages where page and image loads tend to be grouped together. This represents the “light” network load. The second class replaces four of the web-browsing hosts with users who issue requests to an FTP server for a 7MB file every 5 minutes while continuing to browse the web. This represents the “medium” network load. The “heavy” network load raises the number of FTP users to 12 from 4.

A. Simulation Metrics

To evaluate the effects of TRACK on network performance, the following metrics are considered:

1) Ethernet delay

The per-packet Ethernet delay reflects the total time frames spent queued or being forwarded through the Ethernet network. This metric will be impacted both by the TFT processing time, which increases queuing delay, but also by the increased frame size which increases the forwarding delay.

2) Average packet queue length, Mac Relay Entity

The average queue length shows the number of packets, on average, waiting to be processed by the Mac Relay Entity (MRE) in the switch. Since the TFT is part of the MRE, the average number of frames waiting to be serviced by the MRE is expected to increase due to the increased time frames will spend in the forwarding process. The queue length is reported in frames, time-averaged for the entire simulation.

3) Dropped frames

When the queues fill up, frames must be dropped – either upon receipt or from the queue to make room for the new frame. This may cause application or protocol retransmission mechanisms to activate which could make the problem even worse at layer 2. At no point during any of the simulations did the switch queues become full and so no frames were dropped for any scenario.

4) Application Throughput

In the best case, the change in layer-2 protocol is transparent to the higher layers. This metric measures the relative difference in average Bytes/sec transmitted to the host from the server in response to an application request. Results are averaged across all runs of a given scenario.

VI. SIMULATION RESULTS

A. Ethernet Delay

As expected, the Ethernet delay increases as the TRACK processing time increases for the light-load network. The results show a delay of $168.6 \pm 1.8 \mu\text{s}$ for the baseline case. Fast and medium-speed TRACK processing raises the delay to $187.1 \pm 1.5 \mu\text{s}$ and $255.0 \pm 2.0 \mu\text{s}$, respectively. Using the slowest implementation TRACK raises the delay to $590.2 \pm 4.7 \mu\text{s}$.

The larger variation in Ethernet delay values as TRACK processing delay is increased is due mostly to effects of TRACK processing on interpacket arrival times which causes the queuing behavior at the switches to change. While an increase in delay is not desirable, the increase in variance may prove to be more problematic, especially for applications that

require strict quality-of-service maximum delay enforcement. It is significantly easier to guarantee a delay below a certain bound with the fast TRACK implementation than with the slow TRACK implementation.

The delay distribution for the medium-load network both expands and shifts towards the right compared to the light-load network. This indicates that both the delay and delay variance are increasing. These results are shown in Figure 3a.

For the medium-load network with TRACK disabled, frames are subject to an average delay of $199.0 \pm 2.1 \mu\text{s}$ delay. The fastest TRACK implementation raises the delay to $214.5 \pm 1.5 \mu\text{s}$, an increase of about 7%. The medium-speed implementation raises the delay to $278.5 \pm 6.2 \mu\text{s}$, up about 33%. The greatest increase, 102%, occurs with the slow TRACK implementation which causes the delay to be raised to $616.5 \pm 10 \mu\text{s}$. The relative delay increases experienced for these scenarios are roughly the same as the increases experienced for the lightly-loaded network.

The results for the heavy-load network show an average delay of $224.5 \pm 5.1 \mu\text{s}$ for the baseline case. Enabling the fast TRACK implementation causes this to increase to $236.6 \pm 2.5 \mu\text{s}$, an increase of about 5%. The medium-speed implementation causes the delay to increase to $297.3 \pm 3.2 \mu\text{s}$, an increase of about 28%. As expected, the maximum delay increase is experienced with the slow implementation of TRACK which resulted in a delay of $648.3 \pm 12 \mu\text{s}$, an increase of 97%.

B. MAC Relay Entity (MRE) Queue Length

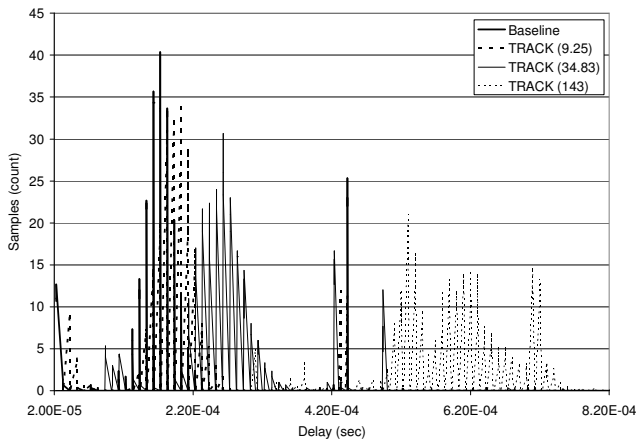
The results show that for the light-load network with TRACK disabled, the queue contained, on average, 3.69×10^6 frames. Fast and medium TRACK increased the average queue lengths to 2.13×10^5 and 7.26×10^5 , respectively. As expected, the largest increase occurred with slow TRACK with an average queue length of 3.80×10^4 . TRACK processing resulted in queue size increases of 141%, 181%, and 196% for the fast, medium, and slow implementations, respectively. At no point did the queue become full, and so no frames were dropped.

The queue length results for the medium-load network are shown in Figure 3b. Overall, the queue lengths were higher for the medium-load network when compared to the lightly-loaded network, as expected. The percent increase was nearly the same or slightly lower. The baseline case had an average queue length of 2.91×10^4 . That increased to 1.32×10^3 frames when fast TRACK was enabled. It increased to 5.06×10^3 and finally to 2.61×10^2 for the medium and slow-speed TRACK implementations. These values represent increases of about 128%, 178%, and 196% over the baseline case.

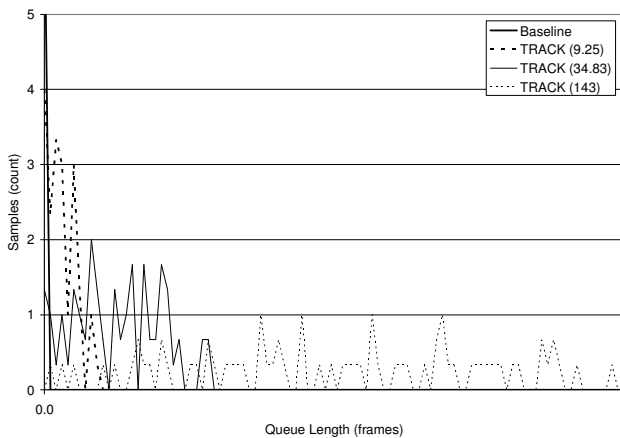
The queue length values for the heavy-load network were about 2 times higher than for the medium-load case. For the baseline case, the average queue size was 6.40×10^4 . For fast, medium, and slow track, the average queue sizes were 2.56×10^3 , 9.78×10^3 , and 5.39×10^2 , respectively.

C. Application Throughput

The results show that the performance impact on applications for all network loads is minimal. The average number of



(a)



(b)

Figure 3 – Ethernet Delay and MRE queue length (for medium network load)

Bytes/sec dropped to no less than 99% of the baseline value when TRACK was enabled in the light-load network. In the medium-load network, the greatest performance drop observed was to no less than 97% of the baseline value. For the high-load network the worst drop was to 93.3% of the baseline value. In all scenarios, the worst performance drop occurred when the slow implementation of TRACK was enabled.

VII. FUTURE WORK

While TRACK provides a reliable DLT mechanism, it contributes to increased network transmission and processing overhead. The primary problem identified with xDGA was that the FDB's at the switches could become out of sync with the data stored on the xDGA [1]. The authors describe this problem in their paper and TRACK was created partly in response to this problem. In a DLT scheme, the event of greatest importance is that a MAC address location is learned. For this reason, it is only necessary to tag frames that cause the FDB at any switch to be updated. Under normal conditions, this occurs relatively infrequently. For frames that are untagged, the TACH could use its previously-learned knowledge of the network topology to map the MAC address to a switch and port. In this case of a topology change, many FDB changes will occur on core switches and the TACH must be aware of the new topology to interpret the new tags correctly. This dynamic protocol should be investigated since it could

allow significant overhead reduction while maintaining the accuracy of the proposed TRACK protocol.

VIII. CONCLUSIONS

Data-Link layer Traceback (DLT) is an important and under-researched problem. This step is important to completing the attack path analysis which would otherwise be missing the critical step between the zombie network edge and the zombies themselves. This paper has presented TRACK, a solution to the DLT problem for wired Ethernet networks. TRACK allows a single packet to be traced back to the switch and port number of the host that sent it. TRACK uses frame tagging to support the DLT operation regardless of the source MAC address used to transmit the frame. It improves upon the reliability of a previous DLT scheme by avoiding the indirect mapping achieved when using FDB information. TRACK's operation is based on principles established by widely accepted and implemented protocols, and TRACK additionally leverages existing standards and protocols to control tagging and collecting operations.

REFERENCES

- [1] H. Hazeyama, M. OE, and Y. Kadobayashi, "A Layer-2 Extension to Hash-Based IP Traceback" in *IEICE Transactions on Information & Systems*. pp. 1-9. Vol. E86-D, No. 11, November 2003.
- [2] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer, "Hash-Based IP Traceback" in *Proceedings of the xth ACM Special Interest Group on Data Communication (SIGCOMM'01) Conference*. pp 3-14. San Diego, CA, August 2001.
- [3] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback" in *Proceedings of the 2000 ACM Special Interest Group on Data Communications (SIGCOMM'00) Conference*. pp. 295-306. Stockholm, Sweden, August 2000.
- [4] D. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback" in *Proceedings of the 2001 Conference on Computer Communications (INFOCOM'01)*. pp. 878-886. 2001.
- [5] H. Burch and B. Cheswick, "Tracing Anonymous Packets to Their Approximate Source" in *Proceedings of the 14th annual USENIX Large Installation System Administration Conference*. New Orleans, December 2000.
- [6] W. T. Strayer, C. E. Jones, B. I. Schwartz, J. Mikkelsen, C. Livadas, "Architecture for multi-stage network attack traceback," in *Proceedings of the 30th Anniversary IEEE Conference on Local Computer Networks*. pp. 15-17. November 2005.
- [7] T. He, L. Tong, "Detecting Encrypted Interactive Stepping-Stone Connections," in *Proceedings of the 2006 IEEE International Conference on Acoustics, Speed and Signal Processing (ICASSP)*. pp. 816-819, May 2006.
- [8] X. Wang, D. Reeves, "Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays" in *CCS'03*. Washington, D.C., October 2003.
- [9] IEEE Std. 802.1Q, "Virtual Bridged Local Area Networks." 2005.
- [10] IEEE Std. 802.1ac, "Frame Extensions for Virtual Bridged Local Area Network (VLAN) Tagging on 802.3 Networks." 1998.
- [11] IEEE Std. 802.1D, "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges." February 2004.
- [12] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-hashing for message authentication," Internet RFC 2401, February 1997.
- [13] *OpNET Modeler*. <http://www.opnet.com/products/modeler/home-2.html>.
- [14] N. Sklavos and O. Koufopavlou, "On the Hardware Implementations of the SHA-2 (256, 384, 512) Hash Functions," in *Proceedings of the 2003 International Symposium on Circuits and Systems*. pp. V-153-V-156. May 2003.
- [15] L. Dadda, M. Macchetti, J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exhibition*. pp. 70-75. February, 2004.