4-19-2016

The user attribution problem and the challenge of persistent surveillance of user activity in complex networks

Claudio Taglienti
Nova Southeastern University, ctaglienti@comcast.net

James D. Cannady Jr.
Nova Southeastern University, cannady@nova.edu

The User Attribution Problem and the Challenge of Persistent Surveillance of User Activity in Complex Networks

Claudio Taglienti[a], James Cannady[a]
[a]*Nova Southeastern University*

Abstract. In telecommunication networks, the user attribution problem refers to the challenge faced in recognizing communication traffic as belonging to a given user when information needed to identify the user is missing. This problem becomes more difficult to tackle as users move across many mobile networks (complex networks) owned and operated by different providers. The traditional approach of using the source IP address as a tracking identifier does not work when used to identify mobile users. Recent efforts to address this problem by exclusively relying on web browsing behavior to identify users, brought to light the challenges of solutions which try to link up multiple user sessions together when these approaches rely exclusively on the frequency of web sites visited by the user. This study has tackled this problem by utilizing behavior based identification while accounting for time and the sequential order of web visits by a user. Hierarchical Temporal Memories (HTM) were used to classify historical navigational patterns for different users. This approach enables linking multiple user sessions together forgoing the need for a tracking identifier such as the source IP address. Results are promising. HTMs outperform traditional Markov chains based approaches and can provide high levels of identification accuracy.

Keywords. Accuracy Scalability, Attribution, Complex Networks, Mobile Networks, Concept Drift

Introduction

The internet of people is becoming the internet of things and it is going to be mobile. Communication devices attached to gas meters, vending machines, fleets of trucks, payment kiosks, as well as, android phones enabled as WIFI routers, ipads, and iphones, all seek, sometimes without requiring human control, persistent connectivity to different resources via complex networks. In this new and dynamically evolving environment it is becoming increasingly difficult to identify these devices and their users.

Complex networks represent graphs with patterns of connectivity that are neither purely regular nor purely random but instead follow a particular mathematical function, known as the power law where these graphs expand continuously with the addition of new vertices and new vertices tend to attach preferentially to other vertices that are already well connected. The hyperlink connectivity of documents in the World Wide Web, the pattern of connectivity of users accessing web documents on the web, the nodes that connect the internet as well as mobile networks that attach to the internet from multiple locations all share the properties of complex networks.

Traditionally, users are identified via authentication techniques which verify the legitimacy of either the user or the device accessing that network.

Once properly authenticated the user/device can access the resources of that network and potentially other networks for which the user had not been authenticated. As mobility is becoming pervasive, users continually move across secured and unsecured networks to access resources available across the internet. A key question that this study has addressed is: "How can users be identified when accessing resources across complex networks when no authentication information is available? The answer to this question has important implications to identification of malicious users re-entering the network. In particular, the traditional user identification problem which leverages authentication to recognize users, morphs into a user attribution problem when user authentication is not possible. In 2010 Clark and Landau [12] acknowledged the need for stronger forms of personal identification that can be observed in the network and defined the attribution problem in terms of a question: "Why don't packets have license plates?". Addressing user attribution allows users to be recognized among many by attributing a trace of past user activity to a given user.

While the academic community has recognized this problem and its complexity, few solutions have been proposed and none address the user attribution problem that ensues when users move across complex networks driven by mobility scenarios that have become a mainstream of personal computing. User identification and user attribution have been addressed in the context of web usage mining [13, 33, 37, 21, 3, 41] but solutions are strongly coupled with the web page structure of specific web sites and cannot be applied in their current form to the more generic user identification problem across multiple web sites accessed via complex networks. More recently "re-identification" has been proposed as an approach, used in dynamic networks like telecommunication networks and the internet, which turns the user identification problem into a matching problem that involves comparing the behavior of network entities such as users across time periods [27]. The re-identification approach has been successfully applied to email-alias detection, author attribution [26] and identification of fraudulent consumers in telecommunication networks, but never in the context of complex networks as defined in this work.

This study makes a contribution to the field of computer information systems by tackling the highly relevant and current problem of user attribution by evaluating the impact of the power law distribution and concept drift present in complex networks. The proposed research has made use of hierarchical temporal memories to record and classify historical user activity in the form of unique time ordered user web site visits. This classification ensures that future user attributions are based on identification of unique patterns of activity that match prior activity patterns by a given user. Hierarchical temporal memories represent a new advance in our understanding of how the neocortex part of a human brain learns and infers sequence patterns over time.

1. The Problem

This research has addressed the challenge that no effective method exists that can recognize the source of communication entering the network or returning to a web site by only utilizing the communication traffic of the user. This problem is further exasperated by the fact that often no form of explicit (user name/password) or implicit (cookies) authentication is available to identify the source of communication. When user authentication is not available, users with their communication traffic can no longer be identified, instead, users can be recognized based on past user activities and the user identification problem can be restated as a user attribution problem.

In order to better appreciate the severity of this problem, consider a malicious user that has been authenticated by an operator network and then proceeds to hack multiple web servers hosted outside the operator network. Imagine then, that this user continues to perform malicious activity while moving between secured and unsecured networks. How can this user be recognized and stopped? Authentication does not help to identify malicious authenticated users if the attack occurs away from the authentication point. In addition, a malicious user can hide his tracks and renew his authentication credentials by switching periodically between network operators. If user authentication cannot effectively be used to identify users re-entering the network then what new approach should be used?

Identification of the source of communication traffic has traditionally relied on the IP address associated with the source of the connection, utilizing it as the client or user identifier. This client identification technique has been used to enforce access-control decisions but suffers from several shortcomings that can potentially make it ineffective [10]:

- A portion of IP addresses are dynamically assigned to clients upon initial connection to the network.
- A portion of IP addresses are allocated behind Network Address Translation (NAT) boxes which hide the real IP address (typically a private IP address) of the client.
- A portion of IP addresses go through web proxies which cause the client IP address to be replaced by a new public IP address

A large number of IP traceback techniques have been proposed to identify the source of communication traffic in the literature as reported by [42, 9, 43, 11, 45]. As pointed out by Santhanam et al [42], most IP traceback schemes are only capable of tracing up to stepping stones (compromised server) which in the context of complex networks, are similar to NATs and Web Proxies in that they assign the source IP address and represent one end point of the communication, thus hiding the real IP address of the user. In addition, individual organizations would find it difficult, if not impossible, to successfully utilize IP tracebacks without the involvement of the upstream internet service provider [6].

As described, traditional security methods that utilize "IP trace back" techniques fail to identify the source of communication associated with users that operate in complex networks (like cellular operator networks) due to the deployment of large cellular gateways that control the source of communication (source IP addresses) for millions of users. Specifically, identification of the source of communication is complicated by the dynamic assignment of source IP addresses to users by these gateways as well as by the presence of large scale NAT and web proxy devices in operator networks. It is difficult to determine how long IP addresses remain allocated to a given user since IP addresses allocated by cellular gateways, out of very large IP pools, persist for longer time periods based on operator configuration (up to 24 hours) than IP addresses modified by NATs or web proxy devices, which are allocated out of much smaller ranges of IP addresses and change very often, typically for the duration of a TCP connection.

1.1. Research Questions

These are the research questions that have provided the original motivation for this study:

- Is it possible to recognize specific users among many in the network by observing and classifying their historical communication behavior and be at least as accurate in the classification process measured using recall as when leveraging comparable classification approaches?
- Does accuracy scale? That is, can the solution maintain the same level of accuracy, as the communication population (number of sources and number of destinations contacted by these sources) increases?

1.2. Assumptions

Two assumptions were made for this study:

(1) HTTP (port 80) traffic was selected as the most representative user communication type traffic since it is used by web browsers which require direct user intervention to navigate. An implication of this study is that it will not be possible to separate any traffic initiated by applications which do not require user intervention/direction but still utilize port 80.

(2) This study assumes that a user uses a single non shared device for all experiments.

1.3. Related Work

The popularity of wireless devices and the rise in supported bandwidth by WiFi and cellular 4G networks has brought to the forefront the user attribution problem in the context of mobility scenarios. Between 2009 and 2013 several researchers have tackled this problem. Two generic frequency based

approaches have emerged from this work. One leverages source IP address based identification to track users and uses frequency of access to visited web destinations to perform inference. Results from this approach are good in terms of accuracy and scalability but accuracy decreases dramatically when the source IP changes. The other approach leverages behavior-based identification which forgoes tracking via the use of a source IP address and only uses frequency of access to visited web destinations. Results are promising in terms of recall accuracy but accuracy does not scale well since frequency of visited web sites does not provide enough unique differentiation among different users especially when few popular web sites dominate test data sets.

In 2009 Kumpošt and Matyáš [32] took on the user attribution problem by leveraging vectors of destination IP addresses bound to specific source IP addresses and classified users based on similarity between train and test data measured using TF-IDF. Experiments results are mixed showing 21% false alarms for SSH, and false alarm rates of 70% and 60% for HTTP and HTTPS. The authors blame the poor results on students moving across campus and getting assigned different IP addresses. In 2010 Herrmann, Gerber, Banse and Federrath [24] use behavior-based identification to tackle the user attribution problem so that users are identified based on access frequencies of web destinations within a fixed user time window which satisfies classification based on a Multinomial Naïve Bayes (MNB) classifier. Experiment results show correct user identification for 50% of the 28 users 80% of the time. Assumption of conditional independence among sites visited limits the scalability of this solution. In 2010, Yang [46] also proposed behavior-based identification. During inference, support and lift are computed to record the strength of patterns of visited web sites followed by calculating the Euclidian difference between learned user patterns and newly inferred ones. Experiment results show 87% accuracy for 100 users using the support based inference. However, the author acknowledges the difficulty of scaling up the number of users due to the inability of the approach to link up consecutive user sessions belonging to the same user. In order to address the scalability problem Yang suggests, as future research, combining behavior-based identification with the use of a tracking identifier like a source IP address. In 2012, Banse, Herrmann, and Federrath [4] use the triplet <epoch, source IP, destination IP> to identify user sessions by aggregating all events that share that same epoch (time frame) and source IP based on a Multinomial Naïve Bayes classifier. Experiment results show correct user identification for 88% of about 2100 user sessions. The authors also acknowledge that changing the source IP address frequently decreases accuracy (60% every 3 hours, 49% every hour). In 2013, Hermann, Banse and Federrath [25] use again the triplet <epoch, source IP, destination IP> to identify user sessions by aggregating all events that share that same epoch (time frame) with a source IP based on a comparison of three classification approaches: 1) 1-Nearest Neighbor Classifier using Jaccard coefficient and Cosine Similarity), 2) Multinomial Naïve Bayes and 3) using

lift and support as proposed in Yang [46]. The best accuracy results record up to 85% recall, using MNB, for over 3000 users with the IP address changing every 24 hrs. Recall accuracy degrades when the source IP address changes frequently (65% every 3 hours, 54% every hour). In Yang's study, behavioral profiling was meant to be used as an additional authentication mechanism (like a behavioral biometric). Therefore, Yang could assume that the learning algorithms will have access to a quite large set of labeled sessions for each user. In fact, the cited result of 87% recall for 100 concurrent users was achieved with 200 training sessions per user (to derive the support-based patterns for the profiles) and 100 test sessions, which were processed as a whole to obtain the support-based profiles, which were to be linked to the training sessions. In contrast, the work by Herrmann et al. [25] links singular sessions.

Previous research on web mining [21, 37, 41] shows that utilizing the source IP is a poor choice for identifying users when the source IP address changes as is the case in mobility scenarios. Previous research also shows that utilizing exclusively behavior based identification does not scale well. To understand why consider using the approach proposed in the literature to identify users that visit 3 popular web sites, say A, B, C. In this case, there exists a single identifiable pattern <A, B, C> distinguishable only based on the user frequency of access of each web site. Now consider recording the order of visits to web sites as an additional way to classify unique patterns. This approach would increase six fold the number of unique patterns :<ABC>, <ACB>, <CAB>,<CBA>,<BAC>,<BCA>. Finally, consider taking into account the time when web sites are visited so that sites visited at approximately the same time represent a single user timed sequence. Now the number of unique patterns increases even more: <ABC>, <ACB>, <CAB>,<CBA>,<BAC>,<BCA>,<A>,<B>,<C>,<AB>,<BC>,<AC>,<CB>, <CA>,<BA>. Increasing the number of unique identifiable patterns helps address the presence of popular web sites in the data set creating conditions for unique differentiation among user patterns that enables to adequately address the user attribution problem.

2. The Approach

The use of timed sequences is at the heart of the approach used in this study to address the user attribution problem. Specifically, variable order Markov chains are used to represent time ordered sequences of web destinations visited by users. States in the Markov chain represent web sites visited and transitions between states represent frequency of visits. Unfortunately, challenges do exist when utilizing traditional Markov chains: (1) Higher Order Markov Chains increase accuracy but decrease coverage [15]. (2) Markov chains incorrectly recognize never learned before sequences [14].

When using Markov chains it is difficult to match many different sequences (high coverage) accurately. The challenge lies in how input sequences are matched against learned input within Markov chains. The learned sequence within a Markov chain matched against the input is known as "*context"*. The most flexible type of Markov chain is the variable order Markov chain where the order (length) of the context is allowed to vary. Variable order Markov chains like PPM-C [35] and All-K [38] attempt to match exactly the input sequence against a context of size N (where N represents the order of the Markov chain). If a match is not found then the input sequence is matched against a shorter context of size N-1, and onward decreasing the size of the learned sequence in the Markov chain until a match is found or a mismatch is declared. In 2004, Deshpande and Karypis [15] have shown that matching a size *N* context increases accuracy but decreases coverage (few sequences are identified accurately), while decreasing the size of *N* upon mismatches increases coverage but decreases accuracy (many sequences identified with lower accuracy). Ultimately it is desirable to achieve both high accuracy and high coverage.

In this study, Markov chain accuracy will be improved using a technique known as state cloning [14] and further extended with a technique known as "Sequence Cloning" introduced for the first time in this study. Consider the Markov chain shown in Fig. 1 created with sequences *abd* and *xbc*. Note that this Markov chain will recognize and generate one of the following four sequences: *abd*, *abc*, *xbd* or *xbc*, where sequences *abc* and *xbd* were never learned.

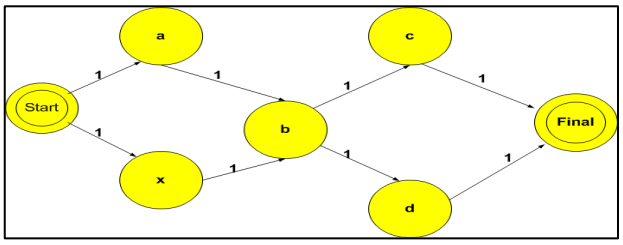


Figure 1 Loss of accuracy in Markov chain

The problem lies with shared state "b" which has the property that its in-degree and out-degree are both greater than 1. When this occurs, the Markov chain will identify more sequences than were learned. To address this problem state cloning (duplicating the shared state) is traditionally used to address the issue as shown in Fig. 2.

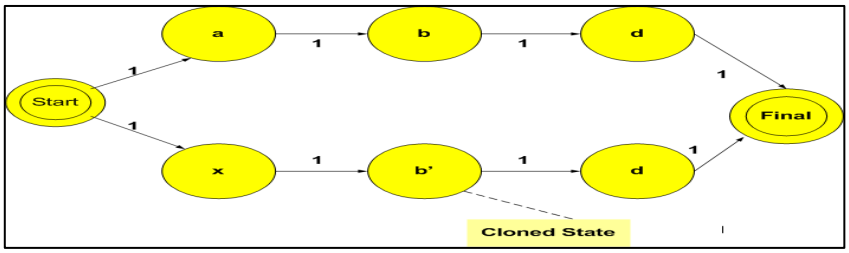


Figure 2 State Cloning

However, traditional state cloning is not always sufficient to address situations were multiple states are shared as shown in Fig. 3.

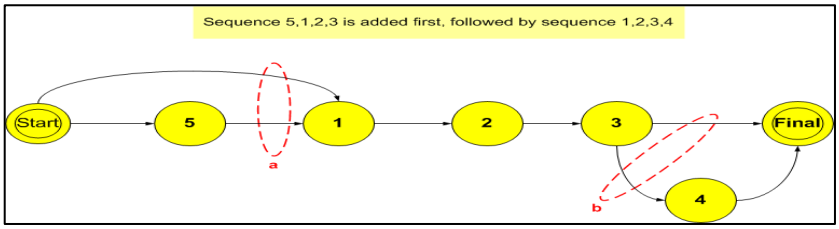


Figure 3 Limitations of State Cloning

The Markov chain in Fig.3 originally learned sequences <5, 1, 2, 3> and <1, 2, 3, 4> yet two more sequences are identified. Note that the single node cloning conditions are not violated, yet this graph produces two sequences that were never learned: 1, 2, 3 and 5, 1, 2, 3, 4. In this case, the problem occurs at the transitions covered by points a and b. These transitions allow the generation through shared nodes 1 and 3 of more than 2 sequences. Namely: <1, 2, 3, 4>, <1,2,3>, <5,1,2,3>, <5,1,2,3,4>. To address this problem the state cloning approach is extended to cover sequences of shared states such that sequence cloning is needed when the first shared node in a sequence of shared nodes has an in-degree greater than 1 and the last shared node in a sequence of shared nodes has an out-degree also greater than 1. By duplicating all shared states we solve the problem as shown in Fig. 4. Note that both state and sequence cloning increase accuracy but also increase the number of nodes in a Markov chain.

This study introduces the idea of accurately matching a learned timed sequence (context) *loosely* not necessarily exactly using a combination of longest common subsequence and longest common substring calculations instead of matching "exactly" the context of learned sequences stored in variable order Markov chains.

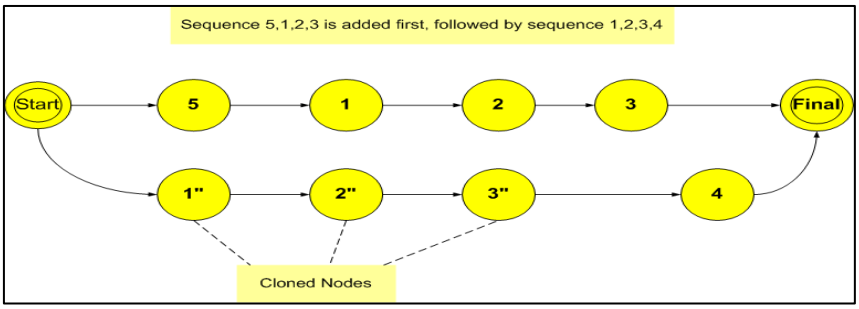


Figure 4 Sequence Cloning

While use of timed sequences of visited web destinations allows increasing the discriminating power of the solution, there still is a need to link up multiple user sessions (sequences) in order to address the poor scalability problem faced by behavioral based identification techniques that forgo the use of a tracking identifier like the source IP address. In this study, hierarchical temporal memories (HTMs) are used to address this need.

A Hierarchical Temporal Memory (HTM) is a technology that is modeled on the algorithms used in the neocortex of the brain [19, 23]. Network nodes in an HTM, are organized in a hierarchical way, with each node implementing learning and memory functions. Hierarchical Temporal Memories are an appropriate tool to study complex networks. HTMs perform well when the data they process support a hierarchical structure. Ravasz and Barabasi [40] show that the scale free and high degree of clustering of complex networks like the World Wide Web are the consequence of a hierarchical organization. They show that a small group of nodes, such as communities of interest in the WWW, organize in a hierarchical manner forming larger groups, while still maintaining a scale free topology. This self-similar nesting of different groups into other groups forces a hierarchical structure that well fits the ability of HTMs to correlate groups that are close in space and time. For more information on why HTMs were chosen to deal with complex networks see Appendix G.

2.1. HTM Inputs

The next few sections will introduce many terms, computations and symbols which are defined in Appendix I. Figure 7 shows a typical HTM with its inputs. Input sequence (*IS*) is the input sequence being matched by a given HTM layer. For HTM layer 1, *IS* is a sequence of web destinations (in a real cellular network these are extracted from HTTP requests) and for HTM layers 2 and 3 it is a sequence of temporal groups (variable order Markov chains) matched in the layer below. At layer 1, input is of the form: *Timestamp<TS, Dest>* where the format of this input is fully described in the "Session Identification" section.

At HTM layers 2 and 3 input is of the form: $\lambda_{Lx} < FFB_{gi}, FFB_{gi+1,} FFB_{gi+2,\ldots} >$ where $2 \leq x \leq 3$ represent layers above the first HTM layer and $1 \leq i \leq$ the number of Markov Chains at HTM layer L_{x-1}. λ_{Lx} is a vector of feed forward beliefs, which measures how well inputs match learned sequences and is a vector where only one index is filled based on which temporal group g_i matched the input at the lower layer L_{x-1} of the HTM. The λ_{Lx} vector represents the input for HTM layers 2 and 3 where the index of the entry in the vector that is filled represents the feed forward belief of the specific temporal group g_i is received from lower HTM layers. For instance, vectors λ_{L2}<0, 0.4, 0> and λ_{L2}<0.8, 0, 0>, show layer 2 of the HTM receiving from layer 1 a sequence of temporal groups S_{g21}(g2, g1) with feed forward belief values of 40% and 80% for temporal groups 2 and 1 respectively.

2.1.1. Session Identification

In this study, the TCP timestamp TS value [29] is used to identify and track user sessions only during the training phase of experiments. The use of TCP timestamps was inspired by the work of Kohno, Broido and Claffy [30]. These authors proposed device recognition by fingerprinting devices via detection of changes in clock skews among different devices using the TCP Timestamp option. Kohno et al., believe that their approach can be used to identify the same physical device among a large number of devices since there exist variability in the clock skew of different physical devices, and it holds that the clock skew for a given device is constant and independent of network access technology. This approach differs from the way in which TCP timestamps are used in this study, where they are leveraged to track directly user sessions and consider clock skew not as a unique fingerprint for devices but instead noise that will decrease the tracking capabilities of this session identification approach. In general, any fingerprinting approach which identifies a logical source (e.g. source IP address) or a physical device as done in Kohno et al. suffers in deployments which obfuscate the source. This becomes particularly problematic in session identification when trying to link up TCP packets belonging to the same user. Consider fingerprinting a device as proposed in Kohno et al. which requires the TCP timestamp value to be passed unchanged through a middle box. Many middle boxes deployed in cellular networks act as "proxies" by splitting the TCP connection between the device and the origin server into two separate TCP connections which cause the origin server to negotiate TCP timestamp options with the middle box instead of the device [28]. This causes the middle box to be mistaken for the real source. This is not a problem when TCP timestamps are used for session identification as proposed in this study. The ability to link up TCP packets belonging to the same user during training will not be impacted whether tapping of communication traffic occurs between device and middle box or middle box and origin server. This is because this approach does not track the source but connections and in this case both connections are surrogates for the single end

to end connection between device and origin server. We are not aware of other work that leveraged TCP timestamps to track communication sessions as was done for this study.

Training of HTMs is completely unsupervised and leverages the tracking strength of the TS value to identify consecutive web visits as belonging to the same user session (observation). This is different from the supervised training approach used by Yang in her experiments where a label (user-id) was used to train her inference model. During training, in this study, session identification and user identification are one and the same. During the inference stage the assumption that a specific session belongs to a given user no longer holds and instead the TS value is only used to identify an anonymous session (a set of consecutive web visits belonging to an unknown user that make up an observation). The task of assigning an anonymous session to a specific user is carried out by the Markov chains performing inference within the different layers of each HTM based on past learned patterns of users' sessions (web visits). All HTMs attempt to recognize each anonymous session and only one HTM will be able to recognize it better than the other HTMs based on its past training.

Beacken et al. [7] have discovered that the TCP Timestamp field used for iphones always starts at the same date/value when the device is restarted but for android devices, the TCP timestamp value on device power up is random. They state that this allows one to be able to distinguish iphones from android type devices. In this study, the TCP time stamp value, a 32 bit value, which implements a virtual clock on each device, is used to uniquely identify unique sessions associated with a given user. The prototype built for this study identifies multiple communication sessions during the training phase of learning that belong to different users by tracking the unique TS value (TS) of each device. During training, all communication input associated with a given <TS> value within a given time window is fed to a hierarchical temporal memory (HTM) to identify the communication patterns associated with sessions belonging to different users. These communication patterns are defined in terms of the destinations (*Dest*, a number mapping to the IP address of the web site) visited by this user. The timestamp, calculated from the input at HTM layer 1, has a resolution of 1 millisecond and represents the passage of time with respect to the arrival of input to the HTM. The time stamp is specifically needed to distinguish multiple <TS, Dest> input pairs immediately following each other with potentially the same TCP time stamp values, as either all arriving at the same time or at different times.

The algorithm in Fig.5 was used to implement communication session identification during the training phase of classification and selection of appropriate HTMs to perform communication pattern identification.

Each HTM_{Ux} once created runs a virtual clock with a 1 ms resolution used to track the TS value of sessions associated with this HTM. The allowed TS clock window was computed as follows: Allowed-TS-Clock-Window = [TS_v +

Clock()] $\pm$ Clock-Skew-Factor. The computation TS_v + Clock() needs to account for wrap around at 2^{32}. The Clock-Skew-Factor is a fixed maximum allowed clock skew.

Unfortunately, using a fixed window offset from the currently received TCP TS counter to measure clock skew, can potentially either underestimate (lose a single tracked user session) the clock skew with a window that is too small or overestimate (identify a single user session as belonging to multiple user sessions) the clock skew with a window that is too large. A possible way to address this problem is to allow for dynamic resynchronization of the HTM TS counter with a tracked source based on how much of an offset (within a window) a given new received TCP timestamp is from the existing HTM TS counter. This approach would use the new TCP time stamp received as the new TS counter value each time the new TS value is within the window but does not match exactly the current HTM TS counter. This could address the potential increase in clock skew that occurs over time overcoming the limitations of a fixed HTM TS counter. With this newly proposed approach, it will be possible to use a small window size since the algorithm is able to adjust to clock skew over time. The benefit of this approach, as well as determining the best size for the clock skew window, is an area of further research that should be based on the empirical results of studying the characteristics of clock skew of mobile devices in real mobile networks.

IF (Given input: <**TS_v**,Dest>, **TS_v** is out of range of allowed TS clock skew window for any **HTM_{Ux}**)THEN
// New user not identified before
// Create a new HTM to track communication patterns from this source
- Create New **HTM_{Ux}** (Timestamp:<**TS_v**,Dest>)
ELSE IF (Given input: <**TS_v**,Dest>, **TS_v** is in range of allowed TS clock skew window for a single **HTM_{Ux}**) THEN
// Existing user already being tracked
- Invoke existing **HTM_{Ux}** (Timestamp:<**TS_v**,Dest>)
ELSE // The TCP timestamp matches more than one HTM
- Drop the input
- Update counter: Unable-to-Distinguish-Session
ENDIF

Figure 5 TCP Timestamp Session Identification Algorithm

2.2. How the HTM Works

Each HTM learns and then performs inference. Learning occurs in an unsupervised manner, starting from the bottom layer of the HTM, one layer at a time. Layer 1 learns first. After that, layer 2 learns and once layer 2 is done learning layer 3 completes learning. During training a new HTM is created for each user each time a not seen before user session (based on TCP timestamp

tracking) is encountered. Learning entails both spatial and temporal learning. Spatial learning at layer 1 covers identification of individual sequences of web destinations, while at layer 2 and 3 it covers identification of individual sequences of coincidences (temporal groups representing Markov chains matched from the layer below).

Initial learning is completed at each HTM layer with creation of a single Markov graph representing all learned sequences within that HTM layer. At the end of training this single graph is split into many variable order Markov chains by merging all nodes that are most highly connected into one of several Markov chains based on a depth first traversal of the Markov graph (see Fig. 6) thus ensuring that sequences are maintained and not broken up and that sequences held in Markov chains do not overlap.

While there are more nodes to be processed from the Markov graph Do

- Pick the next node (seed node) from the Markov graph not yet processed adjacent to the "Start" state. This seed node is the first node of a new Markov Chain g_i
- Perform a depth first traversal of the Markov graph originating from the seed node and add all traversed nodes to Markov Chain g_i that have not been processed yet
- Potentially merge this Markov Chain g_i with another already processed Markov Chain g_x if Markov Chain g_i has elements in common (same node in the Markov graph) with Markov Chain g_x. When merging, smaller Markov chains get merged into larger ones.

EnDo

Figure 6 Algorithm to create Markov Chains from a single Markov Graph

These Markov chains represent destinations or coincidences (temporal groups) that are highly temporally correlated based the specific temporal order in which they follow each other.

After learning is completed at a given HTM layer, playback occurs. During playback, each HTM at layer *Ln* which completed learning is used to bootstrap learning for the layer above *Ln+1* using the already learned sequences at layer *Ln*. Playback (an approach introduced in this study) improves the time it takes to train the HTM and allows higher layers to learn higher level concepts that are consistent with the lower level concepts learned by the layers below. In the playback stage, learned sequences at layer *n* are generated in increasing order of time, so that layer *n+1* can correctly learn higher level concepts from the layer below. This in effect simulates the HTM been retrained on the same input used to train the layer below. In order to generate sequences in increasing time order (from oldest to most recent), each node in the Markov graph holds a FIFO queue of timestamps. Each time stamp represents the time when a node was created or modified by updating or adding incoming or outgoing links

to/from this node. Time ordered sequence generation is achieved by traversing the Markov graph at each layer of the HTM, starting from the "start" state, while removing from the front of the FIFO queues timestamps associated with nodes with the least recent (oldest) timestamp for each transition up to the "final" state.

During inference the HTM layer collects input until a sequence is formed. The spatial and temporal poolers at each layer of the HTM ensure that sequences are created so that nodes that follow each other in space (sequential order of inputs) and time (timely order of inputs) are grouped together creating a sequence matched against learned sequences of coincidences (stored within variable order Markov chains g_i). The HTM spatial and temporal poolers terminate a sequence and start another under one of the following terminating conditions: (TC_1) A fixed maximum input size has been processed. (TC_2) A maximum *learned* inter destinations arrival rate is exceeded. (TC_3) The same destination is already present in the sequence (HTM version 1). Note that only HTM layer 1 uses terminating condition TC_2.

Each HTM layer matches the input sequence collected against learned sequences held in Markov chains and finds the best (longest) matching learned sequence (LLS). Each HTM layer computes the feed forward belief (λ) of the best matching learned sequence (LLS). This feed forward belief combines the degree of membership (how well input matches learned sequences) and persistence (how often a matched learned sequence is visited). Belief propagation occurs when HTM layer *n* passes as input the feed forward belief (λ) to HTM layer *n+1*. Belief propagation for casual (Bayesian) networks was first proposed by Judea Pearl [36] and then adapted to HTMs by Deleep George [19].

The output that is sent to the Max Output Layer (see Fig. 8) from each HTM includes identification of the specific HTM and provides the feed forward belief of the matched observation input across all layers of that HTM. The Max Output Layer aggregates the feed forward beliefs (λ_{Output}) of up to one observation worth of data (50 web sites) from each HTM using one of seven HTM algorithms and then selects the HTM (user) with the maximum aggregated feed forward belief value among all HTMs as the one that best matches the HTM layer 1 input.

HTMs use one of seven algorithms proposed in this study (detailed in section "HTM Algorithms Calculations") to aggregate feed forward beliefs and to determine how well an observation matches HTMs' learned input. Each one of the seven different HTM algorithms (based on the HTM layer where they are applied 1 or 3) combines feed forward beliefs associated with a given input observation based on one of three generic algorithms: average, weighted sum and path probability. The average based algorithm simply computes the averages of feed forward beliefs associated with a given observation. Weighted

sum algorithms (BottomUp and TopTop) use weights proportional to the size of an observation matched by a given feed forward belief. The BottomUp algorithm aggregates the feed forward belief with weights proportional to an observation at layer 1 so that the degree of membership calculation at higher layers will be impacted by this weight as the feed forward belief travels up the HTM layers (see Table 1). The TopTop algorithm aggregates the feed forward belief with weights proportional to an observation matched at layer 3. The Path Probability algorithm leverages the idea of independence among feed forward beliefs and simply multiplies together the feed forward beliefs belonging to a given observation.

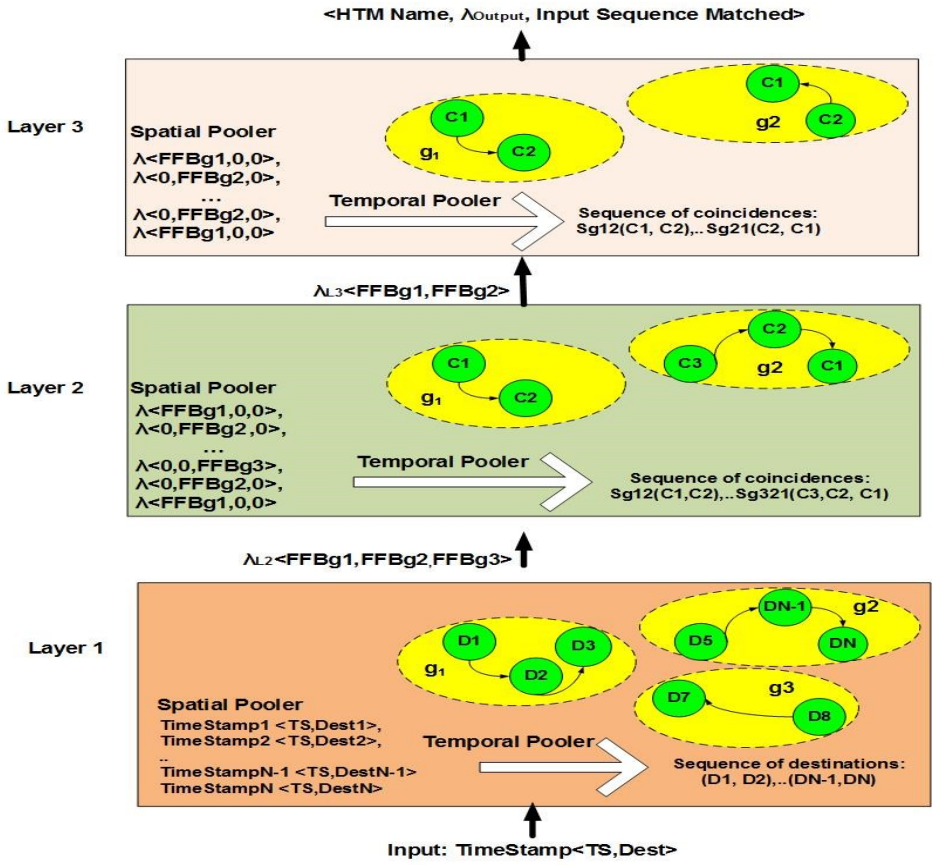


Figure 7 Hierarchical Temporal Memory layers

2.3. Feed Forward Belief Calculations

As shown in Fig. 7 the key computation performed by HTMs creates groups of feed forward beliefs (λ<*FFBs*>) that match the input presented to each HTM layer. The idea is to find the longest learned sequence (*LLS*) out of

all the learned sequences in all of the Markov chains in a given HTM layer which best matches (has the highest degree of membership (*DM*) as shown in equation (8)) the input sequence (*IS*). The calculation of *DM* and persistence (*SP* equation (7)) of the matched *LLS* represents how well inferred input (*IS*) is matched against learned input. In terms of notation: *(Abbreviation) Function name (Arg1,…Argn)* represents a function "*Function name"* which takes *n* arguments *Arg1,…Argn* and is referenced using the abbreviated name "*Abbreviation*". The rest of this section presents the key calculations necessary to compute feed forward beliefs (equations 9 and 10) that are calculated within each HTM layer before being propagated up to next layer. Refer to Appendix I for specific explanations of abbreviations used in this section.

(ALLLS_Cond) Adjusted Length LLS Condition(IS, cLLS) =
if *IS* is a substring of *cLLS* and $IS_1 = cLLS_1$ and $|cLLS| > |IS|$

$$(ALLLS)Adjusted\ Length\ LLS(IS, cLLS) = \begin{cases} |IS| & \text{ALLLS\_Cond} \\ \max(|IS|, |cLLS|) & \text{otherwise} \end{cases}$$

$$(LCSm)\ Longest\ Common\ Subsequence\ Measure(IS, cLLS) = \frac{|LCS(IS,cLLS)|}{ALLLS(IS,cLLS)} \quad (1)$$

$$(LCSUm)\ Longest\ Common\ Substring\ Measure\ (IS,cLLS) = \frac{|LCSu(IS,cLLS)|}{ALLLS(IS,cLLS)} \quad (2)$$

See Appendix C for an example of how to use equations (1) and (2).

(*Fq*) *Frequency of visits* ($cLLS_i$) = Number of times $cLLS_i$ occurs in Markov chain g_i within this HTM layer across all Markov chains in that HTM layer (3)

NLS = Number of all learned sequences within a given HTM layer

$$(Ps)\ Persistence\ (cLLS) = \frac{Fq(cLLS)}{NLS} \quad (4)$$

Similarity Weights = *Ws* + *Wu* + *Wp* = 1.0 (5)
where in this study *Ws* = 0.495, *Wu* = 0.495, *Wp* = 0.01

(SS) Sequence Similarity (IS, cLLS) = (6)
(*LCSm(IS, cLLS)* × *Ws*) + (*LCSUm(IS, cLLS)* × *Wu*)

(SP) Sequence Persistence (cLLS) = (*Ps(cLLS)* × *Wp*) (7)
(DM) Degree of Membership (IS, cLLS) = (8)
max_{LLS}(*SS (IS, cLLS)*) when |∀s, *SS (IS, cLLS)*> *SS(IS,*s) | == 1 (8.a)
max_{LLS}(*SS (IS, cLLS)* , SP(*cLLS*)) when |∀s, *SS (IS, cLLS)*> *SS(IS,*s) | > 1 (8.b)
where s = all learned sequences, including *cLLS*, in a given HTM layer

The best matching *LLS* is that candidate LLS (*cLLS*), in a given HTM layer, that has the maximum degree of membership value when measured against the input sequence (*IS*). For instance, assume IS = <S_1, S_3> which matches $cLLS_1$ <S_1,S_2,S_3,S_4> and $cLLS_2$ = <S_1, S_3, S_6> in Fig.14, then LLS which satisfies DM(IS,cLLS) is $cLLS_2$ since:

$LCS_m(IS,cLLS) = LCS_m(IS, cLLS_2) = 2/2$
$LCSU_m = (IS,cLLS) = LCS_u(IS, cLLS_2) = 2/2$
$LCS_m(IS,cLLS) = LCS_m(IS, cLLS_1) = 2/4$
$LCSU_m = (IS,cLLS) = LCS_u(IS, cLLS_1) = 1/4$
$SS(IS, cLLS_1) = (0.5 \times 0.495) + (0.25 \times 0.495)$
$SS(IS, cLLS_2) = (1 \times 0.495) + (1 \times 0.495)$
$SS(IS, cLLS_2) > SS(IS, cLLS_1)$ then based on (8.a) $max_{LLS}(SS\ (IS, cLLS))$ is $cLLS_2$.

When more than one *cLLS* sequence exists with the same maximum sequence similarity values then the more persistent of these *cLLS* sequences is chosen. Consider a different example where IS = <S_1, S_6> and from Fig.14 $cLSS_1$ = <S_1, S_8, S_6> and $cLSS_2$ = <S_1, S_3, S_6>, in this case equation 8.b is used since two sequences have the same sequence similarity values, namely $|SS(IS,cLLS_1) = SS(IS,cLLS_2)| > 1$. In this case, LLS = <$S_1$, S_3, S_6> since $SP(cLLS_2) = 2/16 > SP(cLLS_1) = 1/16$

(FFB) Feed Forward Belief (IS, LLS) = (9)
min(1.0, DM (IS, LLS) + SP(LLS)) where *min* is the minimum function

FFB(IS,LLS) is the feed forward belief used with all HTM algorithms except for path probability. The feed forward belief applied with the path probability HTM algorithm (see section "HTM Algorithm Calculations" for details) is shown below:

(*FFB_PP*) *Path Probability of LLS* (LLS) = (10)
$P(LLS) = P(LLS_1) \times P(LLS_2 | LLS_1) \times P(LLS_3 | LLS_1\ LLS_2) \times .. P(LLS_n | LLS_1 \ldots LLS_{n-1})$, where:

$$P(LLS_1) = \begin{cases} \frac{Fq(LLS1)}{\text{NLS}} & \text{if } LLS_1 = IS_1 \\ \text{penalty} = 0.0001 & \text{otherwise} \end{cases} \quad (11)$$

where penalty is an error probability for mismatches against *IS*.

$Fq(LSS_{j \to k})$ = Frequency of visits across node transition j→k

$$P(LLS_k \mid LLS_j) = \frac{Fq(LLSj \to k)}{Fq(LLSj)} \quad \text{where } k,j > 1 \quad (12)$$

2.4. Feed Forward Belief Propagation Calculations

Feed forward beliefs propagate through HTM layers, as shown in Fig. 7, following the formulas shown in Table 1. Feed forward beliefs exiting the output (last) HTM layer will be directed to the Max HTM output layer, shared across all HTMs, which aggregates feed forward beliefs for an each observation worth of inputs (see Fig. 8).

Table 1. FEED FORWARD BELIEFS FOR EACH HTM LAYER

HTM Layers	Feed Forward Beliefs Propagation
MAX HTM Output layer Output	$(\boldsymbol{OR})\boldsymbol{Observation\ Result} =$ < HTM_name, *MAX_FFB_HTMs, Observation input*>, *HTM_name* is the name of the HTM matching *Observation input* with the highest aggregated feed forward belief value computed using *MAX_FFB_HTMs* for $S_{\lambda \text{Output}k..k+N}$ over all HTMs.
$HTM_{1..M}$ Layer 3 Output/MAX HTM Output layer Input	$\lambda_{\text{Output}} = \begin{cases} \boldsymbol{FFB\_PP_{gk}(LLS)} \times \boldsymbol{(IAL_{Sgj..j+N})} & \text{if HTM algorithm is Path Probability} \\ \boldsymbol{FFB_{gk}(Sg_{j..j+N}, LLS)} \times \boldsymbol{(IAL_{Sgj..j+N})} & \text{otherwise} \end{cases}$ (13) where $1 \leq k \leq$ Number of Markov Chains at HTM layer 3. Note that when the HTM is configured with only 1 layer then $\lambda_{L2} = \lambda_{\text{Output}}$
$HTM_{1..M}$ Layer2 Output/ $HTM_{1..M}$ Layer3 Input	$\lambda_{L3} = \begin{cases} \boldsymbol{FFB\_PP_{gj}(LLS)} \times \boldsymbol{(IAL_{Sgi..i+N})} & \text{if HTM algorithm is Path Probability} \\ \boldsymbol{FFB_{gj}(Sg_{i..i+N}, LLS)} \times \boldsymbol{(IAL_{Sgi..i+N})} & \text{otherwise} \end{cases}$ (14) where $1 \leq j \leq$ Number of Markov Chains at HTM layer L_2
$HTM_{1..M}$ Layer1 Output/ $HTM_{1..M}$ Layer2 Input	$\lambda_{L2} = \begin{cases} \boldsymbol{FFB_{gi}(IS, LLS)} \times \left(\frac{\boldsymbol{|IS|}}{\boldsymbol{OS}}\right) & \text{if HTM algorithm is BottomUP} \\ \boldsymbol{FFB\_PP_{gi}(LLS)} & \text{if HTM algorithm is Path Probability} \\ \boldsymbol{FFB_{gi}(IS, LLS)} & \text{otherwise} \end{cases}$ (15) where $1 \leq i \leq$ Number of Markov Chains at HTM layer L_1 and Observation size is 50
$HTM_{1..M}$ Layer1 Input received from the network	Timestamp<TS,$Dest_1$> … Timestamp<TS,$Dest_N$>

The input activation level equation shown below measures the strength of a match between input and learned sequences and is further defined in Appendix I.

$$(IAL_{Sgi..i+N})\ \textit{Input Activation Level}(S_{gi..i+N}, \boldsymbol{FFB_{gi}}) = \min\left(1.0, \frac{\sum_{i=1}^{|\boldsymbol{Sgi..i+N}|} \boldsymbol{FFB_{gi}}}{\boldsymbol{|Sgi..i+N|}}\right) \quad (16)$$

2.5. HTM Algorithms Calculations

The seven HTM algorithms described in previous sections are implemented based on the three generic algorithms defined by equations 18, 19 and 20. The algorithms are deployed mainly in the Max Output Layer and also across the HTM layers for the Path probability and BottomUP algorithms. Fig. 8 shows $\boldsymbol{\lambda_{Output}}$ which represents the feed forward belief output from layer 3 of an HTM to the Max Output layer and $S_{\lambda \text{Output} k..k+N}$ which is a sequence of such feed forward beliefs. These parameters are used by the Max output layer to compute the highest valued feed forward belief for a given observation using the formula shown below.

(*MAX_FFB_HTMs*) *Max Feed Forward Belief for HTMs*$(S_{\lambda \text{Output} k..k+N}) =$ (17)

$max_{HTM1..M}$ (HTM_Algorithm$_a$($S_{\lambda \text{Output} k..k+N}$)) where a is one of 3 generic HTM algorithms (Average, Weighted Sum, Path Probability) , $1 \leq M \leq$ number of HTMs/users learned during training, $HTM_{1..M}$ is the name of all HTMs learned during training.

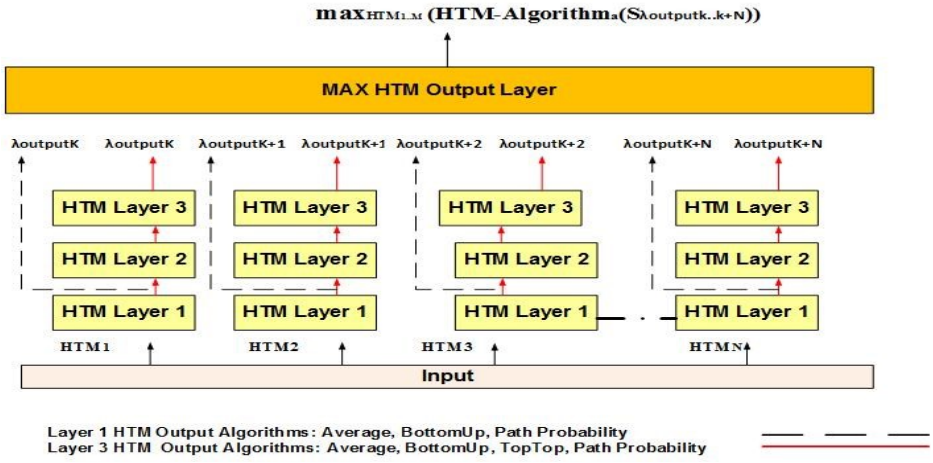


Figure 8 HTM Max Output Layer Inputs and Output

The three generic HTM algorithms which process feed forward beliefs *within* the Max HTM Output Layer are shown below:

$NB = |S_{\lambda \text{Output} k..k+N}|$ is the number of beliefs per observation.

Average $(S_{\lambda \text{Output} k..k+N}) = \left(\sum_{k=1}^{NB} \lambda \text{Output}_k\right) / NB$ (18)

$$\textit{Weighted Sum}\ (S_{\lambda\text{Output}k..k+N},\ IS_k,\ OS) = \sum_{k=1}^{NB}\left(\lambda\text{Output}_k \times \frac{|IS_k|}{OS}\right) \quad (19)$$

where IS_k is HTM layer 1 input sequence associated with λOutput_k and OS = Observation size (50 web destinations at HTM layer 1)

$$\textit{Path Probability}\ (S_{\lambda\text{Output}k..k+N}) = \prod_{k=1}^{NB}\lambda\text{Output}_k \quad (20)$$

See Appendix E for an example on how the Max HTM Ouput layer utilizes HTM algorithms. Table 2 provides more details about which HTM layer provides inputs to the Max Output layer for specific HTM algorithms.

Table 2 HTM Algorithms

HTM Algorithms	HTM Generic Algorithms (HTM_Algorithm$_a$)	HTM Layers supplying input to Max HTM Output
Average	Average	1,3
TopTop	Weighted Sum	3
ButtomUP	Average	1,3
Path Probability	Path Probability	1,3

How to read table 2: The *Average* and *BottomUP* HTM algorithms use the average generic HTM algorithm with inputs to Max HTM Output Layer from HTM layers 1 and 3 (see Fig.8). The *TopTop* HTM algorithm uses the weighted sum algorithm with inputs exclusively from layer 3. Allowing HTM algorithms to generate inputs from either HTM layer 1 or 3 was done to enable verification of HTMs with and without HTM hierarchies. The *BottomUp* algorithm, takes the average of feed forward beliefs at the Max HTM Output Layer computed using the weighted sum at layer 1 (see Table 1 equation (15)). This was done to normalize differences in calculations due to large number of beliefs produced from layer 1 versus the few number of beliefs produced from layer 3 of HTMs.

3. Experiments

The HTM together with the entire set of tools needed to support the experiments conducted in this study were developed from scratch in Java. It became critical then to qualify the HTM to guarantee its correct implementation before performing any experiments. A "calibration" procedure was used to ensure that experiments run using all seven HTM algorithms, as well as, all alternate Markov chain based algorithms would be able to recognize users using their own trained data set instead of a different test data set, achieving in the case of synthetic data sets, 100% recall accuracy results. This initial calibration criterion was used to qualify each one of the HTMs and

alternate Markov chains algorithms as being correctly implemented with respect to their ability to accurately train and infer their own input. It became also important to create reference data sets for the experiments themselves, allowing validation of the algorithms against a well understood baseline. For this purpose synthetic data sets were created (see Appendix F for a description of how synthetic data was created to be as close as possible to real network data). Only if the HTM and alternate Markov chain algorithms could achieve high levels of recall accuracy with these synthetic data sets would a corresponding new set of experiments be conducted with real network data extracted from an operator cellular network. In this study a total of 514 experiments were conducted utilizing synthetic data and 228 experiments utilizing real network data.

Table 3 shows the entire set of experiments conducted in this study. All experiments test the ability of the seven HTM algorithms to attribute communication traffic to users under test. In addition, four alternate algorithms based on Markov chains (MC) were also tested (E_1, E_2, E_6) in order to compare the HTM inference recall accuracy performance against traditional algorithms that, like HTM algorithms, recognize sequences leveraging Markov chains. A single experiment usually entails training either the HTM using one of the seven HTM algorithms or training one of four MC algorithms on a specific data set (synthetic or real) to perform a specific experiment such as user identification. User identification experiments with synthetic data sets were performed without noise (E_1) and with noise introduced in the test data set in the form of either concept drift (E_2) or in the form of DOS or Phish attacks (E_3, E_4). The ability to maintain high levels of recall accuracy performance with increasing number of users (E_1, E_5) and increasing number of destinations (E_1) was also measured. Similar experiments were also performed utilizing data sets collected from a real cellular network (E_6-E_{11}). Experiments E_{13} and E_{14} verify the ability of the HTM to identify users, utilizing real network data, by continuing to learn during the inference phase of the experiment instead of just learning during the train phase of the experiment. Experiments E_{12} measure the accuracy of session identification performed during the train phase of user identification experiments.

Markov chain based algorithms were chosen because of Markov chains' ability to recognize sequences and because the HTM also uses Markov chains, albeit with modifications. The following types of Markov chains (MC) were used to baseline this work: [Fixed Order] 1st Order Markov Chains, [Fixed Order] 3rd Order Markov Chains, [Variable Order] All-K Markov model, where K=3, [Variable Order] (PPM-C) Prediction by Partial Match, where K=3. In this study, mismatches found by MC based algorithms between learned and new input sequences are assigned a fixed penalty when using fixed order Markov chains and a variable penalty proportional to the number of different

destinations matched so far and their frequencies when using variable order Markov chains.

Table 3. FOURTEEN SETS OF EXPERIMENT FOR THIS STUDY

Experiment Types (E_{1-14})	*Data Type*	*Number of Experiments using HTM Algorithms (7)*	*Number of Experiments using alternate MC Algorithms (4)*
(E_1) User Identification no concept drift	**Synthetic**	210	120
(E_2) User Identification with concept drift	**Synthetic**	84	48
(E_3) User Identification under DOS attack	**Synthetic**	21	-
(E_4) User Identification under Phish attack	**Synthetic**	21	-
(E_5) HTM Accuracy Scalability	**Synthetic**	10	-
(E_6) User Identification Alternate MC	**Real**	-	24
(E_7) User Identification HTM2++	**Real**	42	-
(E_8))User Identification HTM2	**Real**	42	-
(E_9) User Identification HTM1	**Real**	42	-
(E_{10}) User Identification under DOS attack	**Real**	21	-
(E_{11}) User Identification under Phish attack	**Real**	21	-
(E_{12}) Session Identification	**Real**	18	-
(E_{13}) User Identification continuous learning	**Real**	12	-
(E_{14}) User Identification under DOS attack with continuous learning	**Real**	6	-

3.1. Experiments Using Synthetic Data

Three hundred and thirty experiments (E_1) were conducted using synthetic data without simulating context drift which used 1000, 5000, and 10,000 visited web destinations. These experiments simulated 5, 20, 50, 100, 500 users accessing the network using 5 days' worth of train data and either one or two days' worth of test data or just 3 observations (one observation "Obs" equals 50 visited web destinations) worth of test data per user. For instance, Table 21 in Appendix H shows that for 5 users and 1000 visited web destinations, 11 experiments are executed using 5 train days and 1 test day (5/1), 11 experiments using 2 days' worth of test data(5/2), 11 experiments using 3 observations (3 Obs) worth of test data for a total of 33 experiments. On the other hand for 500 users using 5000 destinations Table 21 shows 11 experiments being executed using 3 observations in the test data set.

All 132 concept drift experiments (E_2) shown in Table 22 in Appendix H involved visits to 1000 web destinations with 5 users, with 5,10,15,20 training days and 2,3,4,5 test days' worth of test data respectively (see Appendix F for a description of how concept drift was simulated using a random walk algorithm). Concept drift is introduces in the form of 20% new destinations and 10% new

transitions between destinations not in the original train data set. The first line of Table 22 represents the baseline (no concept drift).

In this study, users are identified based on their normal behavior and anomalous traffic is introduced as a form of noise to understand how much user identification accuracy is lost when noise (in the form of an attack) is introduced in the normal communication traffic patterns of users. The idea is not to identify the attack traffic but to identify normal traffic (tied to a specific user) in spite of the presence of embedded attack traffic (noise). The assumption is that when noise is introduced in the form of a phish or DOS attack from the device, it is due to the device having been compromised, possibly based on a download of an infected application. As the subscriber uses his mobile device to browse the internet (normal behavior) the malicious app is at work in the background, launching its phish or DOS attacks. Table 23 in Appendix H shows twenty-one experiments (E_3) which were conducted with HTM algorithms by simulating denial of service attacks, embedded within synthetic network data, where the attack is initiated from individual devices during the test phase to a number of destinations (5, 10, 20) learned at train time. The destinations are attacked repeatedly over time (within a time interval of 5, 10, 20 ms and spaced by a fixed time interval of 5 ms). The idea is to determine how well the HTM can continue to identify users before and after the attack. In these experiments 10 users are used and 4 of them are assumed to be infected and to start DOS attacks during the test phase. The motivation for attacking destinations learned at train time is based on the assumption that perpetrators of DOS attacks typically target sites or services hosted on high-profile web servers such as on-line retailers, banks, credit card payment gateways which are likely to have been visited by the user, thus making the attack less likely to be detected (less conspicuous).

Table 24 in Appendix H shows twenty-one experiments which simulate phishing attacks (E_4), embedded within synthetic network data, which were run using HTM algorithms. For these experiments, attacks are initiated from individual devices during the test phase where unique destinations (1, 3, 5) are randomly selected from *outside* the user training data set (to simulate access to never visited before web phish sites) and attacked within a time interval (1ms, 3ms, 5ms) spaced by a random time intervals (1 minute – 1 hour).

Ten scalability experiments (E_5) were also run using the two best performing HTM algorithms to measure the ability of the HTM to accurately identify users as the number of users increased from 150 to 500 users

3.2. Experiments Using Real Network Data

The next set of user identification experiments used real network data collected from a CDMA/LTE cellular data network in North America over a

period of approximately a month. Experiments were conducted against HTMs using the following parameters: 5 and 10 users, 5 train days and 1 test day, 5 train days and 2 test days, 10 train days and 3 test days. The actual number of different web destinations visited by all users over the month was: 4903 destinations for 5 train days/1 test day, 5221 destinations for 5 train days/2 test days , 6672 destinations for 10 train days/3 test days.

The data originally collected from the network was for 50 users for a period of one month, unfortunately only 10 users used enough communication data to support the train and test timelines proposed for this study.

One hundred twenty six user identification experiments (E_7, E_8, E_9) were conducted using real network data running seven HTM algorithms. Table 25 in Appendix H shows the configuration for these experiments. For instance, 10 users leveraging 5 days of train data and 1 day worth of test data (5/1).

Twenty four user identification experiments were run using alternate Markov based algorithms (E_6). Forty two experiments (E_9) uncovered shortcomings in the HTM state machines when handling repetitive consecutive web destinations embedded in the real network input data set. The HTM was modified to address these shortcomings and the same 42 user identification experiments (E_8), using the same data set, were run to determine if the HTM accuracy could be improved. Finally, repetitive web destinations occurring at the exact same time (same timestamp) were removed from the input data set and the 42 same user identification experiments were run again (E_7).

Twenty one experiments which identify users in spite of simulated DOS attacks (E_{10}) and 21 experiments which identify users in spite of simulated Phish attacks (E_{11}) were also run using real network data (using parameters shown in Tables 23, 24).

The inability to collect TCP timestamps from the real cellular network limited session identification experiments to utilizing synthetically created TCP timestamps. It thus became necessary to conduct a set of experiments (E_{12}) to determine how noise introduced by different session identification algorithms impacts train data and the ensuing inference accuracy of HTMs. Session Identification experiments were performed by creating training data sets for the HTM that use one of three session identification algorithms: (1) Source IP, (2) Sliding Window, (3) TCP Timestamp. The train data set to be modified by the session identification algorithms uses real network data. The experiments include a preliminary step which runs the session identification algorithms against real network data to produce a new altered train data set that is modified based on the bias introduced by each session identification algorithm run under conditions that introduce noise. The experiment would then train the HTM with this altered train data set and use the original real network data as the test data set. Using the “Source IP” algorithm, all input with the same source IP address belongs to the same user. The “Sliding Time Window”

algorithm selects the first (oldest) HTTP request in a time window based on the source IP address and assign it to user-x, then all subsequent HTTP requests within the time window for that source IP address, belong to the same user-x. As long as data is available for user-x within the window over time, then that session belongs to user-x otherwise that session is assigned to a new user (source IP address) selected at random based on users who have data falling within the sliding time window. Note that the sliding window approach presented in other related literature [4, 46] only specified that requests occurring together in time belong to the same session. No other detail was given as to how a specific session was identified among others occurring at similar times. Thus, the use of the oldest source IP in a given time window as the seed for identifying a given user is proposed in this paper in support of this approach. The TCP Timestamp algorithm uses the TCP Time stamp values within a clock skew window to track different users.

A total of 18 session identification experiments were run where the source IP and TCP Timestamps leveraged real life scenarios to alter the original train data set. This was done by random simulation of recycling of the same source IP address among users as done by NATs and web proxy middle-boxes and by random simulation of re-attachment of a device with a new source IP as done when users move across networks. For TCP timestamp, data loss was randomly simulated by creating holes in the data stream as well as random simulation of device power off/on. The number of users in these experiments is 5 and 10 for 5 train days, with the following additional experiment parameters: (Source IP) : 10% recycle source IP address and 10% access network re-attaches; (Sliding Window): Sliding window size in seconds (1, 3, 5, 60); (TCP Timestamp): 10% data loss and 10% device power on/off.

In order to determine if it was possible to further improve HTM user attribution accuracy with real network data in the presence of real concept drift, continuous learning logic was added to the HTM. Continuous learning was implemented by allowing the output of the Max HTM Output layer, which identifies which HTM_x a given observation belongs to, to be sent back to layer 1 of that HTM_x so that it can learn that observation. HTM_x then during inference uses a mechanism similar to "playback" to learn the just received observation across all HTM layers. Two types of continuous learning were implemented: (1) *Continuous Baseline* which lets the Max HTM Output Layer send feedback to the *correct* HTM that matched the given observation. (2) *Continuous Inference* which lets the Max HTM Output Layer send feedback to the *inferred* HTM (which could be right or wrong) that matched the given observation. Twelve user identification experiments (E_{13}) with continuous learning were conducted for 5 and 10 users using the BottomUP approach run at HTM layer 3.

In order to determine if continuous learning could improve the HTM accuracy in spite of DOS or Phish attacks, 6 experiments (E_{14}) were run. These

experiments were conducted using real network data for 10 users using 5 days' worth of train data and 2 days' worth of test data, using the BottomUp Layer 3 HTM algorithm and applying simulated DOS and Phish attacks configured with the same parameters previously described for these experiments in Tables 23, 24.

4. Results

Due to the large number of experiments conducted for this study only key results will be reported in this section. Following are key findings from these experiments:

- HTM algorithms such as Bottom Up and TopTop tend to provide the highest levels of recall accuracy and scalability among all HTM algorithms.
- The HTM algorithm Path Probability tends to perform the worst among all HTM algorithms.
- Alternate Markov chains based algorithms (1st and 3rd Order Markov Chains, All-K and PPM) perform very poorly in terms of recall accuracy and recall accuracy scalability compared to HTM algorithms. However, Alternate Markov Chains based algorithms excel at recognizing their own train input.
- HTM recall accuracy is strongly influenced by the number of visited web destinations and the pattern of behavior (which web sited are visited) of users, specifically:
    - Recall accuracy for HTM algorithms improves dramatically moving from 1000 to 5000 web destinations visited by all users. Beyond 5000 web destinations accuracy levels off.
    - Continuous repetitive patterns (large number of identical web destinations visited over a very short time window) found in real network data impact negatively HTM algorithms' recall accuracy.
    - User behavior changes (new web sites visited) from behavior learned at train time (concept drift) do impact negatively HTM recall accuracy. Concept drift that splits randomly learned sequences of web destinations has the most negative impact on HTM recall accuracy performance.
        - Continuous learning does mitigate the negative impact of concept drift on HTMs' recall accuracy performance.
- Recall accuracy reported by HTM algorithms at layers 1 and 3 is generally comparable.
- HTMs tolerate reasonably well noise introduced in test datasets in the form of DOS or Phish attacks.

- TCP timestamps can be effective in session identification, however if noise is present in the training data set, experiments showed that the sliding window algorithm can be a more accurate session identification algorithm

4.1. Results using Synthetic Data Sets

Based on experiment set (E_1), alternate MC based algorithms with one day worth of test data never produced recall accuracy statistics above 42% (see Table 4 below) and when the test data set consisted of 3 observations, these algorithms never produced recall statistics over 66% (see Table 5). In contrast, HTM algorithms produced accuracy statistics (recall statistics) as high as 99% for a sample of 100 users with one day worth of test data as shown in Table 4 and 99% recall accuracy for a sample of 500 users with 3 observations worth of test data as shown in Table 5.

Table 4 - 5-100 users, 5000 Destinations, 5 Train Days and 1 Test Day Synthetic Data (E1)

(E_1) HTM and Alternate Algorithms	**Variable Number of Observations per user**			
	5 users, 5000 destinations, 5 Train/1Test, Ave Recall	***20 users, 5000 destinations, 5 Train/1Test, Ave Recall***	***50 users, 5000 destinations, 5 Train/1Test, Ave Recall***	***100 users, 5000 destinations, 5 Train/1Test, Ave Recall***
HTM L1 Simple Ave	0.988	0.91	0.918	0.89
HTM L1 Bottom Up	0.99	0.99	0.986	0.96
HTM L1Path Probability	0.99	0.98	0.94	0.92
HTM L3 Simple Ave	0.988	0.91	0.91	0.89
HTM L3 Bottom Up	0.99	0.99	0.98	0.967
HTM L 3 TopTop	1.00	0.997	0.999	0.986
HTM L3 Path Probability	0.718	0.46	0.38	0.298
First Order MC	0.42	0.125	0.0277	0.0139
Third Order MC	0.3579	0.120	0.0277	0.013
All K=3	0.42	0.125	0.0277	0.0139
PPM	0.42	0.125	0.0277	0.0139

Table 5 - 5-500 users, 5000 destinations, for 5 Train days and 3 Observations for test Synthetic Data (E1)

(E_1) HTM and Alternate Algorithms	**3 Test Observations /user**				
	5 users, 5000 destinations, 5 Train/1Test, Ave Recall	***20 users, 5000 destinations, 5 Train/1Test, Ave Recall***	***50 users, 5000 destinations, 5 Train/1Test, Ave Recall***	***100 users, 5000 destinations, 5 Train/1Test, Ave Recall***	***500 users, 5000 destinations, 5 Train/1Test, Ave Recall***
HTM L1 Simple Ave	1.0	0.966	0.886	0.91	0.829
HTM L1 Bottom Up	1.0	0.98	0.97	0.98	0.949
HTM L 1Path Probability	1.0	0.966	0.93	0.946	0.88
HTM L3 Simple Ave	1.0	0.966	0.87	0.91	0.83
HTM L3 Bottom Up	1.0	0.98	0.97	0.976	0.946
HTM L3 TopTop	1.0	1.0	0.99	1.0	0.986

(E_1) HTM and Alternate Algorithms	3 Test Observations /user				
	5 users, 5000 destinations, 5 Train/1Test, Ave Recall	*20 users, 5000 destinations, 5 Train/1Test, Ave Recall*	*50 users, 5000 destinations, 5 Train/1Test, Ave Recall*	*100 users, 5000 destinations, 5 Train/1Test, Ave Recall*	*500 users, 5000 destinations, 5 Train/1Test, Ave Recall*
HTM L3 Path Probability	0.53	0.45	0.366	0.329	0.209
First Order MC	0.66	0.25	0.126	0.069	0.0186
Third Order MC	0.60	0.25	0.120	0.066	0.0186
All K=3	0.66	0.25	0.126	0.069	0.0186
PPM	0.66	0.25	0.126	0.069	0.0186

Why do Markov chains based algorithms perform so poorly in these experiments? Fig. 9 shows, the PPM statistics for the experiments run with 5 users with results shown in Table 5. The percentage of hits and misses were computed for all k orders across all users. The PPM algorithm starts at the highest k order (k=3) and each time the context (input) of size k of the input is not matched the algorithm scales down to a lower k order (matches a shorter portion of the input). Fig. 9 shows that the PPM algorithm operates at k order = 0 about 80% of the time. This means that 80% of the time the PPM algorithm fails to match its input, applies a penalty to the path probability for the input and moves down to a lower k order Markov graph until it reaches k order = 0. This explains the poor performance of PPM and other higher K order algorithms (3rd Order MC, All-K) and also explains why higher order Markov chain algorithms have accuracy performance recall output values similar to lower order Markov chain algorithms.

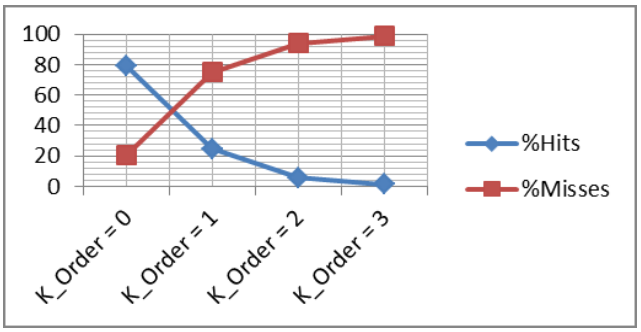


Figure 9 PPM Matches and Miss Matches per K-Order = 3 (E_1)

Accuracy reported by HTM algorithms scales better with increasing number of users and web destinations than the accuracy reported by alternate Markov chains based algorithms as shown in Table 6. Table 6 shows the difference in recall accuracy between experiments(E_1, Table 21 Appendix H) run for 5 and 100 users, with different number of visited web destination (1000, 5000, 10000), over 5 train days and 1 test day using synthetic data. The high and low recall values of all HTM algorithms and alternate algorithms were

recorded and the difference between accuracy values for 5 and 100 users was tabulated. Table 6 shows that HTM algorithms scale better (smaller differences) than alternate algorithms with a maximum of 13% loss in accuracy when tracking 1000 web destinations moving from 5 to 100 users compared to 41% loss in accuracy for alternate MC based algorithms running equivalent experiments. For 5000 and 10,000 web destinations, the scale factor for the HTM algorithm improves even more and is as low as 1% for high recall values.

Table 6 Recall Accuracy Scaling from 5 up to 100 Users for 5 Train Days and 1 Test Day (E_1)

Number of Destinations	**Scale Factor based on Recall Differences from 5 to 100 Users**			
	HTM High Recall Difference	***HTM Low Recall Difference***	***Alternate Algorithm High Recall Difference***	***Alternate Algorithm Low Recall Difference***
1000 Destinations	0.13	0.05	0.407	0.342
5000 Destinations	0.01	0.1	0.41	0.35
10,000 Destinations	0.01	0.06	0.41	0.41

To further understand how accuracy is specifically impacted by the number of destinations in the data set. Synthetic data accuracy performance increases substantially (E_1, Table 21 Appendix H) as number of destinations increases from 1000 to 5000 (from 54% recall accuracy to 99% for 500 users with 5 days of train data and 3 observations of test data), minimally from 5000 to 10,000. Synthetic data accuracy scalability measured for increasing number of users with 5000 web destinations visited (E_5) is high for the top 2 best performing HTM algorithms (BottomUp at layer 1 and TopTop) consistently at 99% recall accuracy for 150, 250 350, 450, 500 users (see Fig. 10).

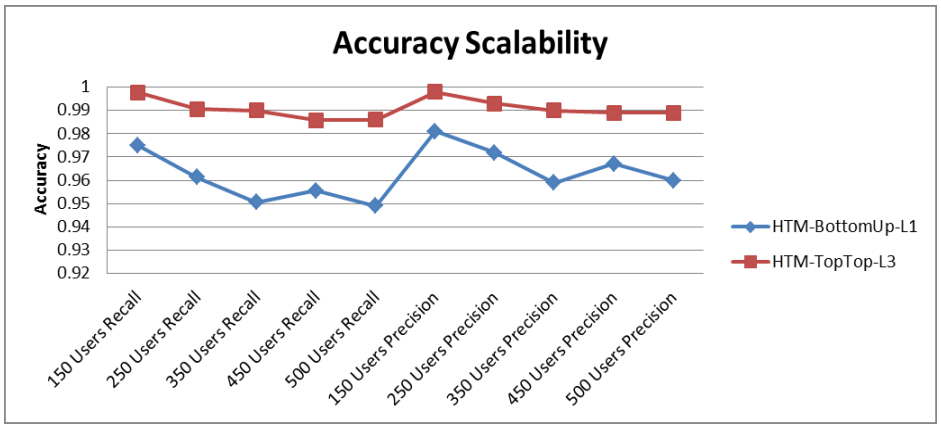


Figure 10 Accuracy scalability Synthetic data, 5 Train and 1 Test Days (E_5)

The weighted sum based HTM algorithms (BottomUp at layers 1 and 3 and TopTop) outperformed all other HTM algorithms. To understand why, let us compare the average based algorithm to the weighted sum based algorithm. Consider an observation of size 5 (5 web sites visited) where the input received by the HTM is <1, 2, 3, 4, 5> which matches 100% of the learned input as 5 *distinct* nodes (5 sequences each of size 1). That is, for the average HTM algorithm FFB= 1+1+1+1+1/5 = 1.0 (100%). Now consider the calculation (as done for the BottomUP algorithm) for the average weighted sum proportional to the matched input FFB = [(1/5) + (1/5) + (1/5) + (1/5) + (1/5)] /5 = 1/5 (20%). Now assume a new input <1, 2, 3, 4, 5> which matches 100% of the learned input as a *single* sequence of 5 nodes (1 sequence of size 5). In this case, using the average algorithm FFB= 1/1 = 1 (100%), while using the average weighted sum algorithm FFB = [5/5]/1 = 1 = (100%). The average algorithm produces the same recall accuracy whether an entire sequence is matched or only individual elements of the sequence are matched, while weighted sum based algorithms give more weight to longer sequences over shorter ones increasing the discriminating power of the solution.

In order to verify the ability of the HTM to handle noise, several experiments were conducted. Experiments (E_2) using synthetic data with simulated concept drift via "Random Walk" show a reduction in accuracy of up to 25%, indicating that the HTM is susceptible to random splitting of learned sequences. Concept drift which adds new connections at the end of learned sequences reduces accuracy only by up to 11%. Experiments which simulated DOS attacks (E_3) run across all HTM algorithms with synthetic data reduced accuracy by up to 11%, while experiments which simulated Phish attacks (E_4) reduced accuracy only by up to 5%.

4.2. Results using Real Network Data Sets

Experiments were also conducted with real network data collected over a period of a month from a cellular data network. Train data sets ranged from 5 to 10 days and 1, 2, 3 test days. Results, at first were modest (E_9). For instance, for 5 users with 5 days' worth of train data and 2 days' worth of test data produced results with recall accuracy as high as 81% for HTM algorithms and 10% for Markov chains based algorithms. Visual observation of this data set showed a high recurrence of repeating continuous patterns of a single destination (e.g. 48, 48, 48, 48) within observations compared to similar measurements for synthetic data. This was confirmed by intra-observation repetitiveness (IOR) measurements which for some users were as high as 94 %, indicating that within a user observation on average there were 47 repeating destinations out of 50. Calibration of real network data run against the HTM also showed poor performance with the best HTM algorithms (ButtomUp and TopTop) scoring recall accuracy values (for experiments E_9 with 5 users, 5

train and 2 test days' worth of data) ranging from 71% to 100%. Unexpectedly, alternate Markov chain algorithms in calibration tests with the same data set performed very well with the lowest recall accuracy value of 99%. It appears that alternate Markov chain based algorithms perform well when test and train data are very similar but when the data set differ as in the user identification experiments (E_6) then recall accuracy scores do not go above 10% .

The HTM was modified (version 2 HTM2++) in the implementation of the sequence termination condition (TC_3) of the HTM spatial and temporal poolers to account for continuous repeated destinations. The same set of experiments was repeated (E_7) but this time any repeated destinations that occurred at the *exact* same time were removed from the real network dataset. The reduction was applied to all train and test data files and accounted for a total reduction in repeated destinations of about 35%. The IOR values decreased and recall accuracy increased. For instance, with real network data for 10 users with 5 train days and 1 test day, IOR decreased by 7% and 9% for train and test data sets respectively while recall accuracy increased by 8% with the recall accuracy values shown in Table 7.

TABLE 7. REAL CELLULAR NETWORK DATA RECALL ACCURACY RESULTS (E7)

Train/Test Days	**Recall Results from Real Cellular Network using HTM algorithms**			
	5 Users High Recall	***10 Users High Recall***	***5 Users Low Recall***	***10 Users Low Recall***
5 Train, 1 Test days	0.95	0.87	0.75	0.64
5 Train, 2 Test days	0.90	0.79	0.78	0.61
10 Train, 3 Test days	0.86	0.81	0.72	0.64

To determine if results using 10 users with real network data had statistical significance we took the experiment from Table 7 using real network data for 10 users for 5 train days and 1 test day. This experiment produced a "high" recall accuracy value of 87% (86.7). We repeated the same experiment 30 times, skipping the first 5, 10, 15, 20,.. up to 150 destinations for each experiment. This mimics starting an experiment at a different place in the real data stream. The value of α chosen was 0.01, the research hypothesis was that the recall accuracy over these experiments was greater than 85.8%. The calculated sample standard deviation was 0.0084, the standard error mean was 0.0015, and the mean was 0.8625, while the z-score was 2.92. These results are significant at the 0.0018 level.

Why was HTM1 (the original version of the HTM) unable to recognize repetitive continuous patters? HTM1 broke up repetitive patterns instead of treating them as sequences. So pattern, 1,2,3 1,2,3 1,2,3 was seen as pattern 1,2,3 occurring 3 times (which is good), but sequence 2,2,2,2,2,2,2,2 was seen as a single destination 2 visited 8 times. This means that a user who seldom visits destination 2 and another who visits it in a sequence will produce

analogous similarity statistics since for a single repeating continuous destination, HTM1 does not see a sequence of destinations but only a single element. HTM version 2 (HTM2) modifies terminating condition TC_3 to continue to process repeated destinations already in the sequence until a new destination not already in the sequence is encountered or terminating conditions TC_1 or TC_2 are met. In order to understand the recall accuracy improvements between experiments E_9, E_8,and E_7, Fig. 11 shows the results for user identification tests run with 5 users with 5 train and 2 test days' worth of real data. HTM1 represents version 1 of the HTMs without the fix to address continuous repetitive patterns (E_9), HTM2 is the second version of the HTM which addresses repetitive continuous patterns but is run on the same real data set as HTM1 (E_8). HTM2++ is HTM2 run on the real data set where web destinations repeated at the exact same time are removed from the input (E_7). The baseline in Fig. 11 is based on running the experiments on the equivalent synthetic data set. The association of higher levels of IOR measurements with the inferior accuracy results was further investigated and experiments were run (beyond experiments reported in Table 3) with the same data set for 10 users (5 train days/1 test day), this time completely eliminating repeating continuous patterns of a single destination within observations in the input to determine if this would further positively impact accuracy results. While IOR continued to decrease (additional 8% for both train and test data sets), unexpectedly, accuracy never improved beyond 87%.

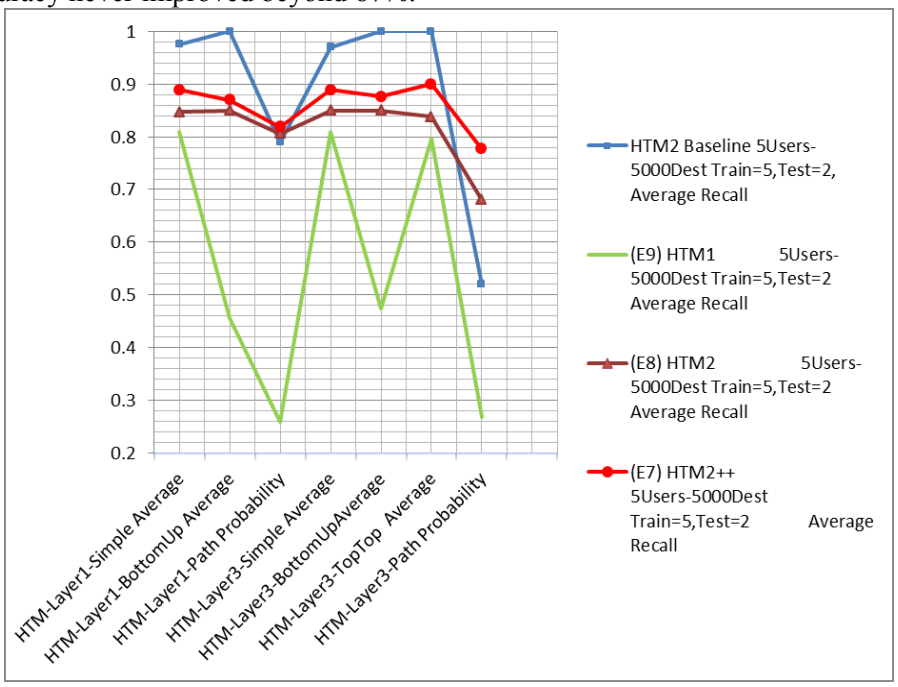


Figure 11 - Accuracy comparisons of all HTM versions (E7,E8, E9) including removal of same time destinations

While investigating these results further, the real network data set was run through an algorithm that computed and thus measured concept drift by converting the concept drift generator for synthetic data to a concept drift detector for real network data. This concept drift detector identified changed user behavior in the test data set due to visits to new connections among existing nodes (CD_e) and new connections to new nodes (CD_n) as shown in Table 8.

TABLE 8. CONCEPT DRIFT MEASURED IN REAL CELLULAR NETWORK DATA (E7)

Train/Test Days	**Concept Drift (CD) from Real Network Data**			
	5 Users CDe existing connections	***10 Users CDe existing connections***	***5 Users CDn new connections***	***10 Users CDn new connections***
5 Train, 1 Test days	0.19	0.20	0.05	0.07
5 Train, 2 Test days	0.20	0.21	0.07	0.09
10 Train, 3 Test days	0.18	0.19	0.07	0.08

The average CD_e concept drift is 20% and CD_n is 7% which is very similar to the simulated concept drift levels used for synthetic data. In order to determine the impact of concept drift on recall accuracy for real network data experiments, the HTM prototype was further modified to *continuously learn* during inference (E_{13}). Preliminary results, using only the BottomUP approach run at HTM layer 3, show that recall accuracy improves with the real network data for 5, 10 users for 5 train days and 1, 2, 3 test days. Specifically, recall accuracy for 5 users improved on average 4% and 2% for continuous baseline and continuous inference respectively and 6% and 1% respectively for 10 users. Table 9 shows details of these results.

TABLE 9. RECALL ACCURACY WHEN CONTINUOUS LEARNING IS APPLIED TO REAL NETWORK DATA (E13)

Train/Test Days	**Continuous Learning (BottomUp layer3) Recall Results from Real Cellular Network**			
	5 Users Recall Continuous Baseline/Normal	***10 Users Recall Continuous Baseline/Normal***	***5 Users Recall Continuous Inference/Normal***	***10 Users Recall Continuous Inference/Normal***
5 Train, 1 Test days	0.983 /0.95	0.931 /0.87	0.983 /0.95	0.899/0.87
5 Train, 2 Test days	0.88/0.876	0.847/0.788	0.868/0.876	0.776/0.788
10 Train, 3 Test days	0.847/0.773	0.826/0.764	0.807/0.773	0.766/0.764

Experiments with real network data which simulated DOS attacks (E_{10}) run across all HTM algorithms reduced accuracy by up to 8% while experiments which simulated Phish attacks (E_{11}) reduced accuracy only by up to 4%. Fig. 12 shows the results of (E14) using continuous learning to mitigate the user attribution effects under a DOS attack. The baseline, "normal" in Fig. 12, represents experiments run without continuous learning. The results show that

when a subset of users (four out of ten) is under attack during the inference phase, it is possible to get improved attribution recognition accuracy (even over scenarios where no attacks are present) when the HTM algorithm can learn perfectly (continuous baseline). On the other hand, using inference to select which sequences to learn during continuous learning (continuous inference) produces mixed results. DOS attacks conducted during the inference phase produce decreased recognition accuracy performance when continuous inference learning is enabled possibly due to the fact that attacked sites were already learned by this user before the attack took place and do not create new distinctive patterns. Phish attacks instead produced better recognition accuracy performance possibly due to the fact that attacked sites were new and not learned until after the attack making it easier to infer them correctly.

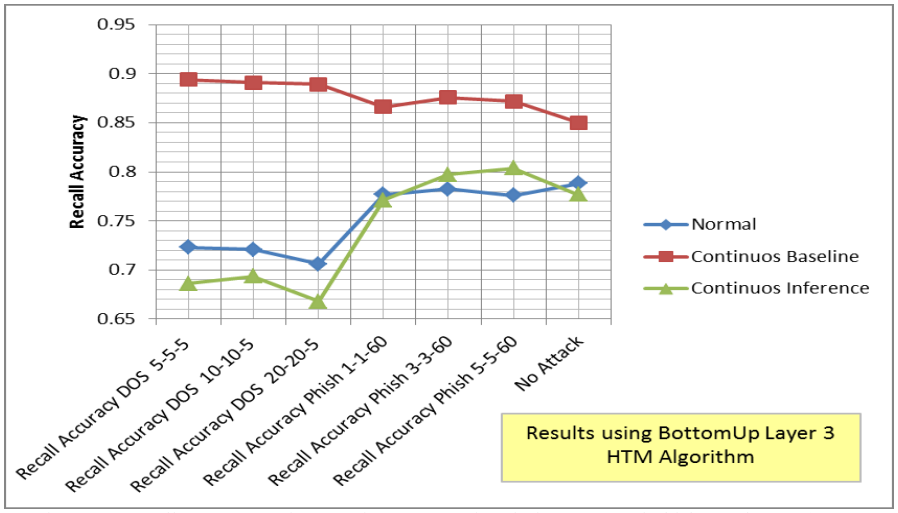


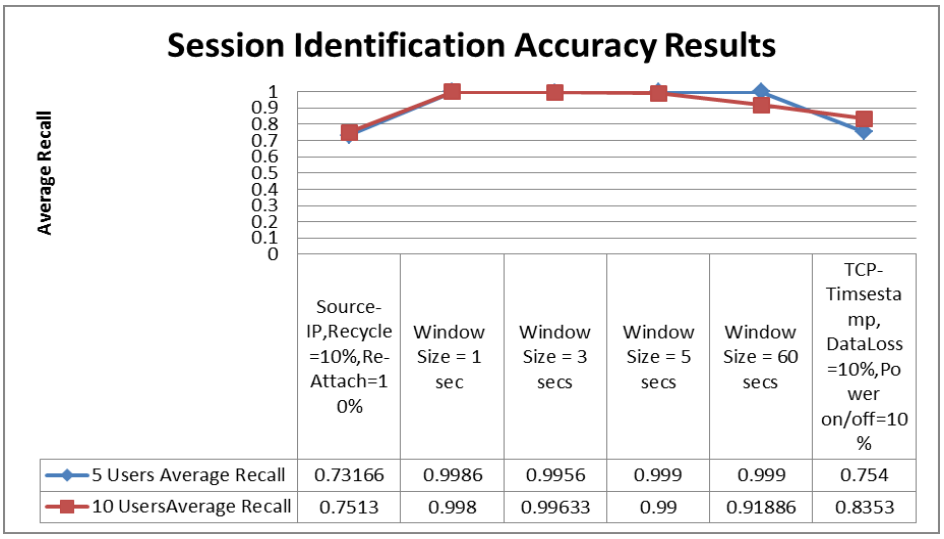
Figure 12 Recall Accuracy using Continuous Learning during DOS and Phish attacks (E14)

4.3. Session Identification Experiment Results

The motivation for conducting session identification experiments is to determine how much different session identification approaches applied during HTM training impact the ability of the HTM to infer accurately (when used under conditions that emulate real life scenarios specific to each session identification algorithm).

Session Identification experiment results showed that the sliding window session identification algorithm (which operates normally with some level of randomness) when measured against "perfect" tracking algorithms such as source IP address and TCP timestamp (that are exposed to simulated real life conditions which introduce noise in the data set) can outperform both of these algorithms. The average recall accuracy for all experiments across 5 and 10 users is shown in Fig. 13. To put things in perspective, a window size of 1 minute, produced a change in the original train data set (loss of web

destinations) of 57%, yet recall accuracy during inference is reported at 92% (sliding window worst result), compared to the best performing TCP timestamp (with 10% data loss and 10% device resets) which reports average recall accuracy value of 83%. These results warrant further study into the impact of noise on the session identification algorithms presented in this study.



	Source-IP,Recycle=10%,Re-Attach=10%	Window Size = 1 sec	Window Size = 3 secs	Window Size = 5 secs	Window Size = 60 secs	TCP-Timsestamp, DataLoss=10%,Power on/off=10%
5 Users Average Recall	0.73166	0.9986	0.9956	0.999	0.999	0.754
10 UsersAverage Recall	0.7513	0.998	0.99633	0.99	0.91886	0.8353

Figure 13 Aggregate Session Identification Recall Accuracy Results

5. Security and Privacy Considerations

This study addresses the important problem of user attribution leveraging communication traffic. A relevant property of user attribution as implemented in this study is that it is privacy preserving with respect to the real identity of the user. Consider the following real-world scenario where the user attribution solution described in this paper is deployed in an access network (possibly a cellular/WIFI operator network or at internet points of presence) and monitors HTTP traffic. As traffic passes through the user attribution solution (UAS) as proposed in this paper, the solution learns to recognize users (User 1, User 2, User 3, …., User N) based on each user's past communication behavior. After the learning stage, the UAS can recognize users (inference stage) based on learned communication patterns when users re-enter the network possibly using a new source IP address and new authorization credentials without knowing the user specific identity (User1 is Joe Smith). In the context of security, accuracy and the (minimal) amount of test data used by HTM algorithms to recognize users are the key measures of success for the UAS. The majority of experiments conducted in this study measured recall accuracy; in addition,

experiments conducted which leveraged just 3 observations in the test data sets provide good insights on how quickly the user attribution solution proposed in this study could recognize users.

There are several applications in the area of security that benefit from being able to address the user attribution problem. Specifically, in the area of intrusion detection, the HTM-based approach used to identify users can be used before, during or after an attack has taken place to identify the communication traffic associated with the user that needs to be stopped or rate limited.

The "user attribution problem" is generic and not tied to attack scenarios, but it can still be used to recognize and stop malicious users. Consider a second real-world scenario where the UAS is coupled with an intrusion detection and prevention system (IDPS) so that both receive the same communication input but instead of using the source IP address to recognize users (due to the unreliability of this source), the IDPS uses the user labels (User 1, User 2, User 3…. ,User N) associated with the given input provided by the UAS. The UAS after it has completed the training phase and it has entered the inference phase provides user labels to the IDPS. Assume the IDPS (out of scope for this study) detects anomalous behavior with communication traffic belonging to user4 and blocks all HTTP traffic associated with this user label. User4, unable to access the internet decides to re-enter the network next day from a different location, possibly using a different access network (e.g. WIFI access point instead of a cellular network). User4 likely using a brand new assigned IP address is able to access the internet, until the UAS recognizes user4 and passes this user label to the IDPS which will again block this user before the user can start to perform malicious activities.

The experiments run in this study show that it is possible to recognize communication traffic as belonging to a given user even during DOS or phish attacks originating from mobile devices albeit less effectively. We are not aware of reported DOS attacks that originate from mobile devices. Operators do report DOS attacks such as DNS amplification attacks originating from the internet destined for IP addresses of mobile devices. Such attacks are typically stopped by operators using firewalls deployed at internet peering points and do not impact user attribution as defined in this study, since they do not originate from mobile devices. Analysis of DOS attack traces made available by the operator where real network data was collected show the typical DOS attack pattern of extremely high volume of messages, sent within extremely tight time windows to the same destination IP address. If such attack patterns were to originate from many mobile devices then the UAS could easily filter them out (same destinations within the same time stamp) as was done in experiments E_7.

What if after an attack occurred, traces of the communication traffic of many users were analyzed to tie that traffic to a possible physical human? The

UAS could be trained to recognize communication traffic for users and bind those users to real humans (user4 is Joe Smith). Like previous examples, the UAS relies on inference to attribute communication traffic to a given user and thus it suffers from possible false negative or false positive errors as opposed to a typical authentication system which relies on fixed tokens like secret key/passwords/cookies. However, it is highly unlikely that such tokens are present in the communication traffic collected from communication logs/traces for user traffic destined to specific web destinations. Even if present, these security tokens have meaning only to the end systems that issued them. Because of these reasons the HTM-based approach proposed in this study could be used to address network forensic investigations.

What if the described scenarios of intrusion detection and network forensic could only have access to encrypted user traffic data? Would the HTM-based approach still be able to recognize user communication traffic? Consider the following three different scenarios: (1) User communication to a secure network server via IPSEC, (2) User communication to a web server supporting a secure HTTPS connection, (3) User leveraging the anonymizing Tor network [16] to communicate with a web server. The answer to the above questions depend on whether the destination IP address in the encrypted communication traffic received by the attribution system represents the actual destination that the user intended to visit. This is *possible* with IPSEC since the IPSEC tunnel could terminate at the site being visited, it is *likely* with HTTPS since the secure connection terminates at the server being visited, but *impossible* with Tor destined traffic since Tor encrypts the original data, including the destination IP address, several times and sends it through a virtual circuit comprising successive, randomly selected Tor relays.

What if a strategic attacker, who knows about the attribution system being in place, takes actions to avoid it? The attribution system monitors normal traffic and not malicious traffic, assuming that the original user has possession of the compromised device, then that user would continue to use the device normally, with normal "concept drift" over time, plus the background malicious activity. In this case the attribution system has a good chance of recognizing the user. However, if the malicious activity completely takes over the device (possibly the device is stolen) then the new malicious user will have different user behavior and this malicious user would not be recognized by the attribution system.

Another security application where user attribution as addressed in this study could be leveraged is in the area of user communication traffic identification under source spoofing conditions. Assume that spoofing or impersonation could take place as two users call or message each other. In this case, the sequence of destinations (numbers called or messaged to) originating from a potential impersonator are run against HTMs which are trained with

sequences of destinations accessed by valid users. These sequences would be identified as being spoofed if the HTM inferred user (associated with these learned sequences) does not match the reported originating user. For instance, assume that for Voice over IP (VOIP) scenarios the HTM at train time monitors SIP INVITE messages which identify calls originating from user1. At train time the HTM learns that user1 calls <user2, user5, user8> in that order. Assume that user1 is being impersonated and the *impersonating* user1 calls <user5, user8, user2>, then this call pattern is not likely to match user1 (the reported originating user in the SIP INVITE message) call patterns and thus user1 would be identified as a possible impersonator. Note that even if the impersonator were to guess the right sequence order of users to call, it is not guaranteed that the HTM would be fooled into identifying the impersonating user1 as the authentic user1. This is because the HTM learns the inter destination arrival rate TC_2 for all learned sequences. This means that if the HTM learned the following sequence <user2, user5, user8> for user1 and for another user4 it learned sequence <user5, user8>, then the impersonating user1 could call the same users as the original user1 but with different inter arrival times so that two sequences could be generated: <user2>, <user5, user8>. In this case user4, different from the originating user1 reported in the SIP INVITE message, would be identified as matching the input sequences and user1 would again be identified as an impersonator. Session identification, however, would likely leverage a different identification approach for the VOIP scenarios than what has been proposed in this study. Assuming that the train data set is trusted, the originating user is identified in the SIP INVITE message ("FROM/Contact" SIP headers) and this tracking header can be used to create user sessions, instead of using the TCP timestamps. This session identification approach works especially well when SIP INVITE messages ride over the UDP protocol which cannot use the TCP timestamp.

6. Limitations

The following sections describe implementation constraints faced during the experimentation phase as well as envisioned challenges faced using the TCP timestamp approach.

6.1. Limitations in Implementation

The HTM prototype was completely written in Java. The performance scalability of HTMs measured in terms of run-time and space (memory needed at run-time) was a challenge in this study which limited the experiments to a maximum of 500 users. All experiments were executed on a Quad i7-3820QM 2.7-3.7 GHz with 16Gig RAM laptop. Threading (one thread per HTM) and caching (of already computed results derived by performing inference traversal of the Markov chains within each layer of the HTM) were two techniques that

considerably improved the inference performance of the HTM allowing completing the experiments for 500 users in reasonable times. When first implemented threads improved run-time performance by almost 100% so that running the HTM algorithms for 2 users would take about 30 minutes to complete in single threaded mode but using multiple threads the experiment would complete in 16 minutes. By adding caching of results of Markov chain searches, performance dropped from 16 minutes to 6 minutes for the same set of experiments. Further optimizations in how threads were used (limiting the number of concurrent threads to 8) and other enhancements in the cache algorithms to maximize cache hits and minimize collisions resulted in run times of 16 minutes to 1 hour and 15 minutes for 5 to 100 users using 5 days' worth of synthetic train data and 1 day worth of test data and 7 minutes to 5 hours for 5 to 500 users for 5000 destinations using 5 days' worth of synthetic train data and 3 observations worth of test data. In contrast, the highest run time for alternate Markov chain based algorithms was 3 minutes for equivalent tests which involved 500 users. Alternate Markov chains algorithms have much better run times since discovery of the start of the input sequence is determined in constant time and matching of the sequence against the "context" occurs in time proportional to the size of the input since all alternate Markov chain algorithms are based on extensions of a 1st order Markov graph which matches the input completely based on the on the very first web destination in the sequence. HTM algorithms on the other hand have search run times that are proportional to the size of the entire input (all sequences) learned at training time as well as the size of the input sequence since HTMs perform an exhaustive search of all Markov chains at each HTM layer.

The amount of runtime memory needed by HTMs also proved to be a limiting factor in being able to extend experiments beyond 500 users. The HTM was run with a JVM setting of 14 gigabytes of RAM but a limiting factor of the HTM design is the need for the MAX HTM Output layer (as shown in Fig. 8) to receive and hold one observation's worth of feed forward beliefs from each HTM before being able to decide which HTM has the "best" feed forward belief. Increasing the number of users increases the number of HTMs which also increases the amount of RAM main memory needed to run the experiment. When the Java JVM starts to run out of the allocated RAM memory and starts to use hard drive virtual memory, run-time performance deteriorates dramatically eventually preventing forward progress.

6.2. Limitations in Approach

There are a few limitations tied to the use of TCP timestamps as a communication session identification algorithm. In this study, the TCP timestamp session identification is used in two situations. The first occurs during training of an HTM when session identification is used to identify up to 50 observations as belonging to a given source associated with a specific HTM

representing a given user. The second occurs during inference when session identification is used to create anonymous sessions for each observation which are distributed to all HTMs so that each HTM can perform inference on the observation to determine how well the observation matches learned input for each HTM. *During the training phase* the following scenarios are not allowed as they will corrupt the training data set:

- A single user leveraging multiple devices
- Many users sharing the same device
- The user recycling the device.

A single user using multiple devices at training time would appear as many different sources and thus one HTM would be created for each source. This is a problem because each HTM created at train time identifies a different user when actually training is occurring for only one user. When many users share the same device at train time, the sessions produced are corrupted. These sessions will include data from multiple users and would thus not be representative of any given user. When a user recycles his/her device, the TCP timestamp is reset and this user session will appear as a new source and thus would incorrectly cause a new HTM to be created, thus identifying a new, non-existing user. During the inference phase when anonymous sessions are created only the second scenario is disallowed as the multiple users sharing the same device will create individual sessions with corrupted data belonging to multiple users. During the inference phase a user can use one device to train the HTM and a different device to perform inference. In addition, during inference, when anonymous sessions are created, the device under test can be recycled, as long as partial sessions (containing less than 50 observations) produced because recycling interrupts creation of the previous observation, are discarded. In general, the effect of a recycling device is the restart of a TCP IP connection which resets the TCP timestamp's TS value. TCP connections are also reset when TCP connections go through middle boxes (Web proxies, NATs, Firewalls, Load Balancers) which split a single TCP connection from device to origin server into two TCP connections; one connection from the device to the middle box and the other from the middle box to the origin server.

If the TCP approach has the reported limitations why was this approach chosen for this study? The TCP timestamp approach provides a reliable way to "track" user communication sessions in a way that is independent of the location or network a device attaches to, thus enabling support for mobility. The authors believe that in real life scenarios, users recycle their mobile devices infrequently, and do not share their personal mobile devices with others. Another practical challenge with the use of TCP timestamp is due to the possibility of loss of tracking accuracy due to clock skew, approaches to address these challenges were presented in the "Session Identification" section. A final limitation brought to bear in this study is found in experiments (E_{14}) where DOS attacks during user identification experiments are countered by

using “continuous learning”. The results show consistent worse recall accuracy performance when using continuous inference learning. This leads to the observation that continuous learning can improve user identification recall accuracy in the presence of “concept drift” but when presented with repeated continuous patters found in typical DOS attacks, it can perform poorly.

6.3. Further Research Directions

An assumption made in this study is that users can be identified by learning web sites they visit. An interesting extension of this idea would be to study the user attribution problem in the context of peer-to-peer communication as is the case for messaging and voice calls. A key question would be: Can users be identified based on peer-to-peer patterns of communication? A related question would be: Is the power law at work in peer-to-peer communication scenarios as well? That is, do people tend to communicate (message or call) with a few set of users very often and with many other users infrequently? The security implications of extending the user attribution problem to cover peer-to-peer communication scenarios are especially relevant in the area of detection of spoofing of sources in peer-to-peer communication (e.g. recently RFC 7375 “Secure Telephone Identity Threat Model” has defined voice attacks where the calling party can be impersonated by an attacker).

As mobility will continue to dominate our future and as the internet of people becomes the internet of things, the user attribution problem will eventually morph into a device/user attribution problem. It would be interesting to run experiments that utilize all communication traffic originating from a device, not just HTTP traffic, to extend the attribution problem from a user to a source (device/user).

Due to the promising results achieved in this study it is important to extend experiments that seek to improve recall accuracy utilizing a larger real-network data set studying further the impact on user identification accuracy of techniques like continuous learning. The HTM framework will be made available to researchers who wish to extend this work by contacting the authors.

7. Conclusions

The user attribution problem is an old problem that just recently has received the attention of the research community. This problem is very important in the field of security since if one cannot attribute communication traffic to a specific user in a network then one cannot identify the user to determine if that user is performing malicious activities in that network or even more importantly stop/prevent the user from continuing to perform such activity. This is the first study that addresses the user attribution problem in the context of complex networks where mobility is dominant using an extensive set

of experiments which used both synthetic and real network data. The approach leveraged behavior based identification using HTMs to extend the research of Herrmann and Yang [25, 46]. This research, acknowledges the limitations of traditional tracking identifiers such as cookies and source IP addresses, and introduces TCP timestamps as a new session identification algorithm used to identify communication sessions for mobile users. Results from the experiments conducted in this study are promising. HTMs outperform traditional Markov chains based approaches and can provide high levels of identification accuracy using synthetic data with 99% recall accuracy for up to 500 users and good levels of recall accuracy of 95 % and 87% for 5 and 10 users respectively when using cellular network data. Performance was further improved with recall accuracy results of 98% and 90% for 5 and 10 users respectively by implementing continuous learning enabled during inference within HTMs to address the challenge of concept drift found in real cellular networks.

This research has made several contributions in the approach used by extending the hierarchical temporal memory model originally proposed in [19] which was not designed to support sequences and showed that sequence based hierarchical memories can consistently provide higher levels of identification accuracy with higher levels of accuracy scalability than traditional Markov chains. The following represent contributions from this research designed to improve HTM inference accuracy:

- This study implements sequence inference using a novel technique which combines traditional variable order Markov chains with the use of longest common subsequence and longest common substring coupled with the persistence of learned sequences to support seven new HTM inference algorithms.
- This study introduces the concept of *sequence cloning* to improve the learning and inference accuracy of Markov chains and of HTMs.
- This study introduces the concept of *playback* to distribute accurately learned sequences from lower to higher layers of the HTM. This reduces learning times and improves inference accuracy in hierarchical models like HTMs.

This study also provides insights into the impact to recall accuracy of the power law distribution at work in complex networks which creates high levels repetition of popular web sites in the data set. The impact to recall accuracy of noise in the data set was studied in the context of simulation of malicious activities and the simulation and observation of context drift in a real network. Experiment results suggest that while partial elimination from the data set of continuous repetition of popular web sites improves accuracy results, complete elimination of such repetition does not produce further improvements. Instead, addressing concept drift found in real networks shows promise as an area of further research for improving attribution accuracy performance.

8. References

[1] A. L. Adamic. and B. A Huberman, The Nature of Markets in the World Wide Web. *Quarterly Journal of Electronic Commerce, 1,* 2000a, 5-12.

[2] A. L. Adamic. and B. A. Huberman, Power-Law Distribution of the World Wide Web. *Science, 287,*2000b*,* 2115.

[3] P. Baldi, P.Frasconi and Smyth, P. *Modelling the Internet and the Web*: *Probabilistic Methods and Algorithms*. West Sussex, England: JohnWiley & Sons, 2003.

[4] C. Banse, D. Herrmann, and H. Federrath, "Tracking Users on the Internet with Behavioral Patterns: Evaluation of Its Practical Feasibility". Information Security and Privacy Research, 27th IFIP TC 11 Information Security and Privacy Conference Heraklion, Greece: Springer Berlin Heidelberg. 2012, Vol 376, pp. 235-248.

[5] A. L. Barabasi and R. Albert, Emergence of Scaling in Random Network. *Science Journal*, *286*(5439), 1999, 509-512.

[6] A. Belenky and N. Ansari, On IP Traceback. *IEEE Communications Magazine, 41*(7), 2003, 142-153.

[7] M. Beacken, L.Braun, D. J.Imbesi, L. G. Greenwald, M. J. Geller, A . Hartman, A., … and D. Bishop, LGS' government communications laboratory and research for the U.S. government. *Bell Labs Technical Journal: Vertical Markets*, *16*(3), 2011,5-28. doi: 10.1002/bltj.20519

[8] B. Bobier, *Handwritten Digit Recognition using Hierarchical Temporal Memory*, (2007). Unpublished manuscript, Department of Computing and Information Science, University of Guelph, Ontarion, Canada. Retrieved from http://arts.uwaterloo.ca/~cnrglab/?q=system/files/SoftComputingFinalProject.pdf

[9] H. Burch and B. Cheswick, Tracing Anonymous Packets to Their Approximate Source. *Lisa '00 Proceedings of the 14th Conf. Systems Administration,* , 2000, pp. 319-328. Berkeley, CA: USENIX Association Berkeley.

[10] M. Casado and M. J. Freedman, "Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification". Proc. 4th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI). Cambridge, MA: USENIX Association Berkeley. 2007, pp. 173-186

[11] H. Y.Chang, R. Narayanan, S. F. Wu, B. M. Vetter, X. Wang, M. Brown, …. and F. Gong,. Deciduous: Decentralized Source Identification for Network-Based Intrusions. *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management,* 1999, pp. 701-714. Boston, MA: IEEE Computer Society.

[12] D. D. Clark and S. Landau, "Untangling Attribution" in Proceedings of a workshop on Deterring CyberAttacks: Informing Strategies and Developing Options for U.S. Policy. Washington D.C., USA: The National Academies Press. 2010, pp. 25-40.

[13] R.Cooley, B. Mobasher and J. Srivastava, Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems,* , 1999, *1*(1), 5-32.

[14] G. V. Cormack and R. N. S. Horspool, Data Compression Using Dyanamic Markov Modelling. The Computer Journal, 30(6), 1987, 541-550.

[15] M. Deshpande and G. Karypis, (2004). Selective Markov Models for Predicting Web-Page Accesses. ACM Transactions on on Internet Technology, 4(2), 163-184

[16] R.Dingledine, N. Mathewson and P. Syverson. Tor: The Second-Generation Onion router. *Proceedings of the 13th conference on USENIX Security Symposium*, 2004.

[17] J. V. Doremalen and L. Boves, Spoken Digit Recognition using a Hierarchical Temporal Memory. *9th Annual Conference of the International Speech Communication Association,* ,2008,(pp. 2566-2569. Brisbane, Australia: ISCA.

[18] W. Duch, R. J. Oentaryo and M. Pasquier, “Cognitive architectures: where do we go from here?”. Proceedings of the First conference on Artificial General Intelligence Memphis, TN: IOS Press Amsterdam. 2008, pp. 122-136

[19] D. George and B. Widrow. “How the brain might work: A hierarchical and temporal model for learning and recognition”. Stanford University. Dissertation Abstract International, 69(04), 177. (UMI No. 3313576) ,(2008).

[20] J. Gray, P. Sundaresan, S. Engler., K. Baclawski, & P. J. Weinberger (1994). Quickly generating billion-record synthetic databases. *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data* (Vol. 23(2), pp. 243-252). Minneapolis, MN: ACM.

[21] M. Grčar, USER PROFILING: WEB USAGE MINING. *Proceedings of the 7 th International Multiconference Information Society,* 2004, pp. 75−78. Ljubljana, Slovenia: Jožef Stefan Institute.

[22] K. Greff, “Extending Hierarchical Temporal Memory for Sequence Classification”. (Master Thesis, Technische Universität Kaiserslautern AG Wissensbasierte Systeme, Saarbrücken, Germany). 2010. Retrieved fromhttp://www.dfki.de/lt/publication_show.php?id=5462

[23] J. Hawkins, D. George and J. Niemasik. (2009). Sequence memory for prediction, inference and behavior. Philosophical Transactions of the Royal Society B Biological Sciences, 364(1521), 1203-1209.

[24] D. Herrmann, C. Gerber, C. Banse, and H. Federrath. “Analyzing Characteristic Host Access Patterns for Re-identification of Web User Sessions”. 15th Nordic Conference on Secure IT Systems (NordSec). Espoo, Finland: Springer. 2010, pp. 136-154

[25] D. Herrmann, C. Banse, and H. Federrath, Behavior-based Tracking: Exploiting Characteristic Patterns in DNS Traffic. Computers & Security. Journal Computers and Security, Vol 39, 2013,17-33

[26] S. Hill and F. Provost, “The Myth of the Double-Blind Review? Author Identification Using Only Citations”. SIGKDD Explorations, 2003, 5(2), 179-184.

[27] S. B. Hill, D. K. Agarwal, R. Bell, and C. Volinsky, (2006). Building an Effective Reppresentation for Dynamic Networks. Journal of Computational and Graphical Statistics, 15(3), 584–608.

[28] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In ACM/USENIX Internet measurement conference (IMC), pages 181-194. ACM, 2011.

[29] V. Jacobson, R. Braden and D. Borman, TCP Extensions for High Performance Network Working Group. (Report No. RFC 1323). 1992. Retrieved from the Internet Engineering Task Force (IETF) website: http://www.ietf.org/rfc/rfc1323.txt

[30] T. Kohno, A. Broido and K. Claffy, Remote physical device fingerprinting, *IEEE Transactions on Dependable and Secure Computing, 2*(2), 2005, 93-108

[31] P. Kumar and P.R. Raju, (2010). A New Similarity Metric for Sequential Data. International Journal of Data Wharehousing and Mining. 6(4), 16-32

[32] M. Kumpošt, and V. Matyáš. “User Profiling and Re-identification: Case of University-Wide Network Analysis”. Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1-10

[33] B. Liu, Web Data Mining Exploring Hyperlinks, Contents and Usage Data. Berlin, Germany: Springer-Verlag, 2008.

[34] W. J. C. Melis, S. Chizuwa and M. Kameyama, Evaluation of Hierarchical Temporal Memory for a Real World Application. *Fourth International Conference on Innovative Computing, Information and Control,* 2009, pp. 144 -147. Kaohsiung, Taiwan: IEEE Computer Society.

[35] A. Moffat, (1990). Implementing the PPM Data Compression Scheme. IEEE Transactions on Communications, 38(11), 1917-1921.

[36] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 2nd ed. San Francisco, CA: Morgan Kaufmann. 1988, pp143-194

[37] J. Pitkow, In Search for Reliable Usage Data on the WWW. *Proceedings of the Sixth International WWW Conference,* 1997, pp. 451-463. Santa Clara, CA: Georgia Institute of Technology.

[38] J. Pitkow, and P. Pirolli, Mining Longest Repeating Subsequence to Predict world wide web surfing. Proceedings. of USITS' 99: The 2nd USENIX Symposium on Internet Technologies & Systems. Boulder, CO: USENIX Association. 1999, Vo.l 2, pp. 13-13

[39] D.J de S. Price. (1976). A general theory of bibliometric and other cumulative advantage processes. Journal of the American Society for Information Science, 27(5), 292-306.

[40] E. Ravasz and A.L. Barabasi. (2003). Hierarchical Organization in Complex Networks. Physical Review E Journal, 67(2), 1-7.

[41] M. Rosenstein,What is Actually Taking Place in Web Sites: E-Commerce Lessons from Web Server Logs. *ACM Conference on Electronic Commerce,* , 2000, pp. 38-43. Minneapolis, Minnesota: ACM.

[42] L. Santhanam, A. Kumar, and D. P. Agrawal, Taxonomy of IP Traceback. *Journal of Information Assurance and Security*, 2006, *1*, 79-94.

[43] S. Savage, D. Wetherall, A. Karlin and T. Anderson, Practical Network Support for IP Traceback. *Proceedings of the ACM SIGCOM 2000, IEEE/ACM Trans. Networking,* ,2001, pp. 295-306. Stockholm, Sweden: ACM.

[44] C. Song. S. Havlin and H. A. Makse, Self-Similarity of Complex Networks. *Nature*, *433*(7024), 2005, 392-395.

[45] R. Stone, CenterTrack: An IP Overlay Network for Tracking DoS Floods. *Proceedings of the 9th Usenix Security Symposium,* 2000, Vol. 9, pp. 15-15. Denver, Colorado: USENIX Association.

[46] Y. Yang, Web user behavioral profiling for user identification. Decision Support Systems, 49(3), 2010, 261-271

9. Appendix A – Example of HTM creation during the Training Phase

The examples in this section and in Appendix B assume that contiguous repetitive sequences (e.g. <48,48,48,48>) are treated as multiple sequences of size 1 based on terminating condition TC_3 (e.g. <48>,<48>,<48>,<48>) which reflects the implementation in version 1 of the HTM (HTM1 in experiments E_9 found in Table 3).

In order to get a better idea of how beliefs propagate up the HTM network layers, this section of the paper shows what happens during playback of input learned in layer 1 of the HTM as represented in the Markov graph and Markov chains shown in Fig.14 in Appendix B. Input received at layer 1 by the spatial pooler is organized into sequences with the temporal pooler computing corresponding feed forward beliefs as shown in Table 10. Note that during playback the feed forward belief vector only indicates the matched temporal group (in contrast with the inference phase where feed forward beliefs also record the degree of membership value).

Table 10 Learned Input Sequences at HTM Layer 1 with generated FFBs

Sequences Learned	**Feed Forward Beliefs λ<g1,g2,g3,g4,g5,g6>**
S1,S2,S3,S4	λ<0,g2,0,0,0,0>
S1,S2,S5	λ<0,g2,0,0,0,0>
S1,S3,S6	λ<0,g2,0,0,0,0>
S1,S3,S6	λ<0,g2,0,0,0,0>
T1,T2,T3	λ<0,0,0,0,g5,0>
T1,T3,T5	λ<0,0,0,0,g5,0>
T6,T5,T7	λ<0,0,0,0,0,g6>
S3,S7,S6,S1	λ<0,0,g3,0,0,0>
H-L1,H-L2	λ<g1,0,0,0,0,0>
H-L1	λ<g1,0,0,0,0,0>
H-L1,H-L3	λ<g1,0,0,0,0,0>
H-L1	λ<g1,0,0,0,0,0>
H-L1, H-L2, H-L3	λ<g1,0,0,0,0,0>
UL1, UL2, UL3, UL4	λ<0,0,0,g4,0,0>
S2,S6,S5	λ<0,g2,0,0,0,0>
S1,S8,S6	λ<0,g2,0,0,0,0>

After layer 1 completes initial training, layer 1 starts playback of learned sequences towards layer 2. The spatial pooler at layer 2 maps feed forward beliefs from layer 1 into sequence of coincidences using the sequence termination rules $TC_{1\text{-}3}$ previously described to combine input into sequences as shown in Table 11.

Table 11 HTM Layer 2 learned Coincidences

Feed Forward Beliefs from Layer 1	Coincidences	Sequences of Coincidences for Layer2
λ<0,g2,0,0,0,0>	g2 ➔ C_1 (new coincidence)	C_1
λ<0, g2,0,0,0,0>	C_1	C_1
λ<0, g2,0,0,0,0>	C_1	C_1
λ<0, g2,0,0,0,0>	C_1	
λ<0,0,0,0,g5,0>	g5 ➔C_2 (new coincidence)	C_1, C_2
λ<0,0,0,0,g5,0>	C_2	
λ<0,0,0,0,0,g6>	g6 ➔ C_3 (new coincidence)	
λ<0,0,g3,0,0,0>	g3 ➔ C_4 (new coincidence)	
λ<g1,0,0,0,0,0>	g1 ➔ C_5 (new coincidence)	C_2, C_3, C_4, C_5
λ<g1,0,0,0,0,0>	C_5	C_5
λ<g1,0,0,0,0,0>	C_5	C_5
λ<g1,0,0,0,0,0>	C_5	C_5
λ<g1,0,0,0,0,0>	C_5	
λ<0,0,0,g4,0,0>	g4 ➔ C_6 (new coincidence)	
λ<0,g2,0,0,0,0>	C_1	C_5, C_6, C_1
λ<0,g2,0,0,0,0>	C_1	C_1

Having completed initial learning, layer 2 then would convert the received coincidences into the Markov Graph and Markov chains as shown below in Fig. 15. Assuming that initial learning is completed at layer 2, layer 2 starts playback in order to train layer 3 as shown in Table 12.

Table 12 Learned Input Sequences at HTM Layer 2 with generated FFBs

Sequences Learned	Feed Forward Beliefs λ<g1,g2,g3>
C_1	λ<g1,0,0>
C_1	λ<g1,0,0>
C_1	λ<g1,0,0>
C_1 C_2	λ<g1,0,0>
C_2, C_3, C_4, C_5	λ<0,g2,0>
C_5	λ<0,g2,0>
C_5	λ<0,g2,0>
C_5	λ<0,g2,0>
C_5, C_6, C_1	λ<0,0,g3>
C_1	λ<g1,0,0>

Finally, the spatial pooler at layer 3 converts feed forward beliefs received from layer 2 into sequence of coincidences using the terminating condition rules previously described to combine coincidences into sequences of coincidences as shown in Table 13. Fig. 16 shows layer 3 Markov chains.

Table 13 HTM Layer 3 Learned Coincidences

Feed Forward Beliefs	Coincidences	Sequences of Coincidences for Layer3
λ<g1,0,0>	g1 ➔ C_1 new coincidence	C_1
λ< g1,0,0>	C_1	C_1
λ<g1,0,0>	C_1	C_1
λ< g1,0,0>	C_1	
λ<0, g2,0>	g2 ➔ C_2 new coincidence	C_1, C_2
λ<0,g2,0>	C_2	C_2
λ<0,g2,0>	C_2	C_2
λ<0,g2,0>	C_2	
λ<0,0,g3>	g3 ➔ C_3 new coincidence	C_2, C_3
λ< g1,0,0>	C_1	C_1

10. Appendix B – Examples of HTM Inference calculations and FFBs propagation

In order to understand how feed forward beliefs propagate through HTM layers during inference consider the following example. Fig.14 represents Markov chains at layer 1 of the HTM created during the training phase based on the following input sequences (see Table 10 in Appendix A) received in this order during inference: <S1,S2,S3,S4>, < S1,S2,S5>, < S1,S3,S6>, < S1,S3,S6>, < T1,T2,T3>, < T1,T3,T5>, < T6,T5,T7>, < S3,S7,S6,S1>, < H-L1,H-L2>, < H-L1>, <H-L1,H-L3>, < H-L1>, < H-L1, H-L2, H-L3>, < UL1, UL2, UL3, UL4>, < S2,S6,S5>, <S1,S6,S8>. These sequences were collected by the spatial pooler based on the sequence terminating conditions $TC_{1\text{-}3}$ (for this example the max allowed sequence size is 4) previously defined. During the learning phase a single Markov graph is created which is then split into multiple Markov chains based on the algorithm shown in Fig.6.

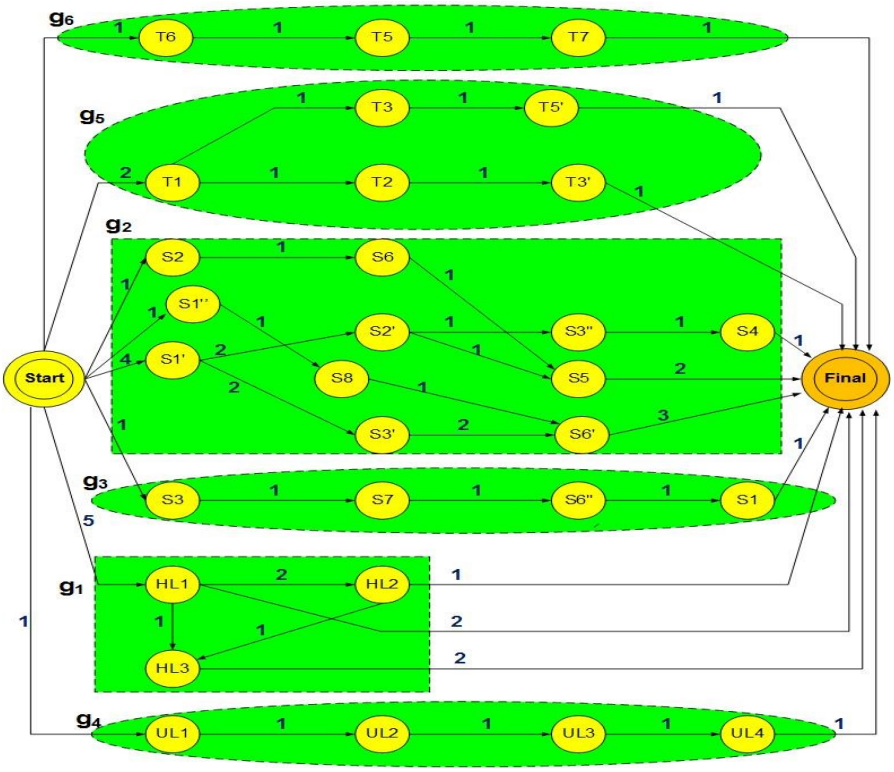


Figure 14 Example HTM Layer 1 Markov Chains

Assume that during the inference stage the following input sequences are processed in time order by layer 1 of the HTM: <S1,S3>, <S3,S6>, <T1,T2,T3>, <T6,T9>, <S7,S1>, <HL1,HL2,HL3>, <UL1, UL2, UL4>, <UL1,UL3>, <S1, S2>, <S1>. Table 14 was created from these inputs based on the FFB calculations from Table 1 applied against the Markov chains in Fig.14. Also assume that the HTM algorithm used is neither the Path probability nor the ButtomUP algorithm.

Table 14 Feed Forward Belief calculations for HTM Layer 1

Input Sequence(IS) to HTM Layer 1	Sequence Similarity (SS)	Sequence Persistence (SP)	Degree of Membership (DM)	λ_{L2}	Markov Chain Matched
<S1,S3>	0.99	0.00125	0.99	<0,0.99,0,0,0,0>	g2
<S3,S6>	0.66	0.00125	0.66	<0,0.66,0,0,0,0>	g2
<T1,T2,T3>	0.99	0.000625	0.99	<0,0,0,0,0.99,0>	g5
<T6,T9>	0.33	0.000625	0.33	<0,0,0,0,0,0.33>	g6
<S7,S1>	0.37	0.000625	0.37	<0,0,0.37,0,0,0>	g3
<HL1,HL2,HL3>	0.99	0.000625	0.99	<0.99,0,0,0,0,0>	g1
<UL1, UL3, UL4>	0.62	0.000625	0.62	<0,0,0,0.62,0,0>	g4
<UL1,UL3>	0.37	0.000625	0.37	<0,0,0,0.37,0,0>	g4
<S1, S2>	0.99	0.000625	0.99	<0,0.99,0,0,0,0>	g2
<S1>	0.99	0.00125	0.99	<0,0.99,0,0,0,0>	g2

The calculations below refer to Table 14 as applied at HTM layer 1 to the Markov chains in Fig.14.

- IS = <S1,S3> then LLS = <S1,S3,S6>, SS = (1×0.495) + (1×0.495) = 0.99, SP = (2/16 × 0.01), DM=SS=.99, λ_{L2} = FFB = min(1,0.991) = 0.99
- IS = <S3,S6> then LLS = <S1,S3,S6>, SS = (2/3 × 0.495) + (2/3 × 0.495) =0.66, SP = (2/16 × 0.01), DM = SS=0.66, λ_{L2} = FFB = min(1,0.661) = 0.66
- IS = <T1,T2,T3> then LLS = <T1,T2,T3>, SS = (1×0.495) + (1 × 0.495) = 0.99, SP = (1/16× 0.01), DM= SS=0.99, λ_{L2} = FFB = min(1, 0.991) = 0.99
- IS = <T6,T9> then LLS = <T6,T5,T7>, SS=(1/3 × 0.495) + (1/3 × 0.495) = 0.33, SP = (1/16× 0.01), DM=SS=0.33, λ_{L2} = FFB = min(1, 0.331) = 0.33
- IS = <S7,S1> then LLS = <S3,S7,S6,S1>, SS= (2/4 × 0.495) + (1/4 × 0.495)= 0.37, SP = (1/16× 0.01), DM=SS=0.37, λ_{L2} = FFB = min(1, 0.371) = 0.37
- IS = <HL1,HL2,HL3> then LLS = <HL1,HL2,HL3>, SS= (3/3 × 0.495) + (3/3 × 0.495) = 0.99, SP = (1/16× 0.01), DM=SS=0.99, λ_{L2} = FFB = min(1, 0.991) = 0.99
- IS = <UL1, UL3, UL4> then LLS = <UL1, UL2, UL3,UL4>, SS= (3/4 × 0.495) + (2/4 × 0.495) = 0.617, SP = (1/16 × 0.01), DM = SS=0.617 , λ_{L2} = FFB = min(1,0.6176) = 0.62
- IS = <UL1,UL3> then LLS= <UL1, UL2, UL3,UL4>, SS= (2/4 × 0.495) + (1/4 × 0.495) = 0.37, SP = (1/16 × 0.01), DM = SS=0.37, λ_{L2} = FFB = min(1,0.371) = 0.37
- IS = <S1, S2> then LLS = <S1,S2,S5>, SS= (2/2 × 0.495) + (1/2 × 0.495) = 0.99, SP = (1/16 × 0.01), DM = SS=0.99, λ_{L2} = FFB = min(1,0.991) = 0.99
- IS = <S1> then LLS = <S1,S3,S6>, SS= (1/1 × 0.495) + (1/1 × 0.495) = 0.99, SP = (2/16 × 0.01), DM =(SS,SP)=0.99, λ_{L2} = FFB = min(1,0.991) = 0.99

The feed forward beliefs from layer 1 travel to layer 2 as shown in Table 15 below. The mapping of temporal groups g_i to coincidences C_i was established during the learn phase and is shown in Table 11 in Appendix A. Layer 2 input sequences (coincidences) are created based on terminating conditions TC_1 and TC_3.

Table 15 HTM Layer 2 Mapping of Feed Forward Beliefs to Coincidences

λ_{L2} λ<g1,g2,g3,g4,g5,g6>	**Maps to Layer 2 Coincidence**	**Layer 2 Input Sequence (IS)**	**Sg_i**	**Layer1 $FFBg_i$ from λ_{L2}**	**Layer 2 $IAL_{Sgi..i..i+N}$**
<0,0.99,0,0,0,0>	g2 ➔C_1	< C_1>	Sg_2	0.99	0.99
<0,0.66,0,0,0,0>	C_1			0.66	
<0,0,0,0,0.99,0>	g5 ➔ C_2			0.99	
<0,0,0,0,0,0.33>	g6 ➔ C_3			0.33	
<0,0,0.37,0,0,0>	g3 ➔ C_4	< C_1, C_2, C_3, C_4>	Sg_{2563}	0.37	0.5875
<0.99,0,0,0,0,0>	g1 ➔ C_5			0.99	
<0,0,0,0.62,0,0>	g4 ➔ C_6	< C_5, C_6>	Sg_{14}	0.62	0.805
<0,0,0,0.37,0,0>	C_6			0.37	
<0,0.99,0,0,0,0>	g2 ➔C_1	< C_6, C_1>	Sg_{42}	0.99	0.68
<0,0.99,0,0,0,0>	C_2	< C_2>	Sg_2	0.99	0.99

The HTM layer 2 input activation level (IAL) calculations from Table 15 are shown below:

- $IAL_{Sg2} = 0.99/1 = 0.99$
- $IAL_{Sg2563} = [0.66 + 0.99 + 0.33 + 0.37]/4 = 0.5875$
- $IAL_{Sg14} = [0.99 + 0.62]/2 = 0.805$
- $IAL_{Sg42} = [0.37 + 0.99]/2 = 0.68$
- $IAL_{Sg2} = 0.99/1 = 0.99$

Layer 2 of the HTM was created during the training phase (see Table 11 in Appendix A) and is shown below.

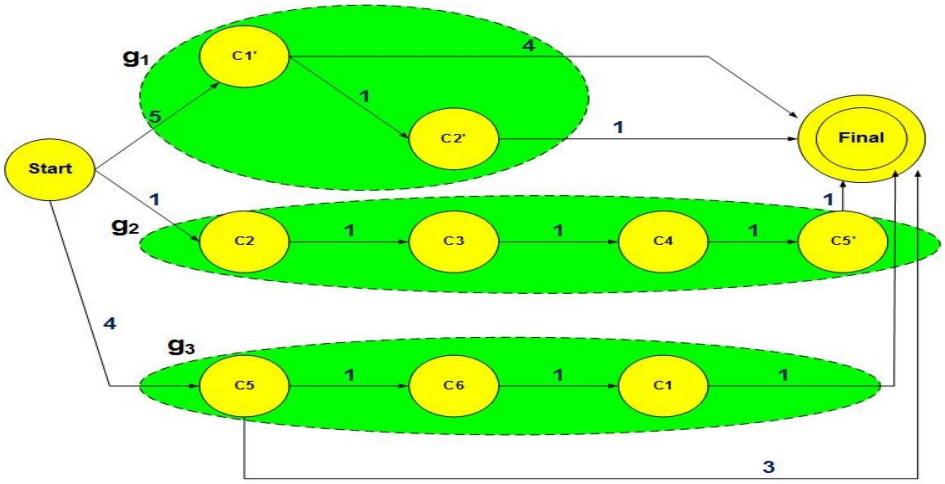


Figure 15 Example HTM Layer 2 Markov Chains

The feed forward beliefs generated at layer 2 to be sent to layer 3 are computed as shown in Table 16.

Table 16 Feed Forward Belief calculations for HTM Layer 2

Input Sequence(IS) to HTM Layer 2	Sequence Similarity (SS)	Sequence Persistence (SP)	Degree of Membership (DM)	λ_{L3}	Markov Chain Matched
$< C_1 >$	0.99	0.004	0.99	<0.98,0,0>	g1
$< C_1, C_2, C_3, C_4 >$	0.74	0.001	0.74	<0,0.43,0>	g2
$< C_5, C_6 >$	0.99	0.001	0.99	<0,0,0.80>	g3
$< C_6, C_1 >$	0.66	0.001	0.66	<0,0,0.45>	g3
$< C_2 >$	0.99	0.001	0.99	<0,0.98,0>	g2

Table 16 above was created based on the following calculations applied against the Markov chains in Fig.15:

- IS = <C1> then LLS = <C1>, SS = (1×0.495) + (1×0.495) = 0.99, SP = (4/10 × 0.01), DM=SS=0.99, λ_{L3} = FFB × IAL = min(1,0.994) × 0.99 = 0.98

- IS = <C1,C2,C3,C4> then LLS = <C2,C3,C4,C5>, SS = (3/4 × 0.495) + (3/4×0.495) = 0.74, SP = (1/10 × 0.01), DM=SS=0.74, λ_{L3} = FFB × IAL = min(1,0.741) × 0.5875 = 0.435
- IS = <C5,C6> then LLS = <C5,C6,C1>, SS = (2/2 × 0.495) + (2/2 × 0.495) = 0.99, SP = (1/10 × 0.01), DM=SS=0.99, λ_{L3} = FFB × IAL = min(1,0.991) × 0.805 = 0.796
- IS = <C6,C1> then LLS = <C5,C6,C1>, SS = (2/3 × 0.495) + (2/3 × 0.495) = 0.66, SP = (1/10 × 0.01), DM=SS=0.66, λ_{L3} = FFB × IAL = min(1,0.661) × 0.68 = 0.449
- IS = <C2> then LLS = < C2,C3,C4,C5>, SS = (1 × 0.495) + (1 × 0.495) = 0.99, SP = (1/10 × 0.01), DM=SS=0.99, λ_{L3} = FFB × IAL = min(1,0.991) × 0.99 = 0.98

Layer 3 of the HTM was created during the training phase and is shown below.

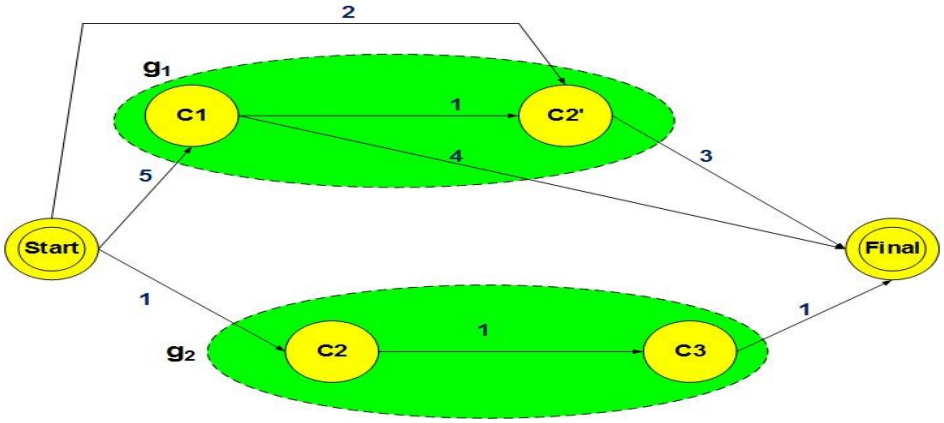


Figure 16 HTM Layer 3 Markov Chains

The feed forward beliefs from layer 2 travel to layer 3 as shown in the Table 17 below. The mapping of temporal groups g_j to coincidences C_j was established during the training phase and is shown in Table 13 in Appendix A.

Table 17 HTM Layer 3 Mapping of Feed Forward Beliefs to Coincidences

λ_{L3} λ<g1,g2,g3>	Maps to Layer 3 Coincidence	Layer 3 Input Sequence (IS)	Sg_j	Layer 2 ($FFBg_i$ × $IAL_{Sgi..i..i+N}$) from λ_{L3}	Layer 3 $IAL_{Sgj..j+N}$
<0.98,0,0>	g1 ➔C_1	C_1		0.98	
<0,0.43,0>	g2 ➔C_2	C_2		0.43	
<0,0,0.80>	g3 ➔C_3	< C_1, C_2,C_3>	Sg_{123}	0.80	0.737
<0,0,0.45>	g3 ➔C_3	C_3		0.45	
<0,0.98,0>	g2 ➔C_2	< C_3, C_2>	Sg_{23}	0.98	0.715

The HTM layer 3 input activation level (IAL) calculations from Table 17 are shown below:

- IAL_{Sg123} = [0.98 × 0.43 × 0.80]/3 = 0.737
- IAL_{Sg23} = [0.45 + 0.98]/2 = 0.715

The feed forward beliefs generated at layer 3 to be sent to the Max Output layer are computed as shown below.

Table 18 Feed Forward Belief calculations for HTM Layer 3

Input Sequence(IS) to HTM Layer 3	Sequence Similarity (SS)	Sequence Persistence (SP)	Degree of Membership (DM)	λ_{Output}	Markov Chain Matched
< C_1, C_2,C_3>	0.66	0.001	0.66	0.49	g1,g2
< C_3, C_2>	0.495	0.002	0.495	0.355	g1

The calculations below belong to table 18 based on the Makov chains of HTM layer 3 in Fig.16.

- IS = <C1,C2,C3> then LLS = <C1,C2> and <C2,C3> since DM(C1,C2) = DM(C2,C3) and SP(C1,C2) = SP(C2,C3), SS = (2/3×0.495) + (2/3×0.495) = 0.66, SP = (1/10 × 0.01), DM=(SS,SP)=0.66, λ_{Output} = FFB × IAL = min(1,0.661) × 0.737 = 0.49
- IS = <C3,C2> then LLS = <C2> since DM(C2) = DM(C1,C2) = DM(C2,C3) and SP(C2) > SP(C1,C2) and SP(C2,C3), SS = (1/2×0.495) + (1/2×0.495) = 0.495, SP = (2/10 × 0.01), DM=(SS,SP)=0.495 , λ_{Output} = FFB × IAL = min(1,0.497) × 0.715 = 0.355

At layer 3 λ_{Output} is sent directly to the Max Output layer, but for layers 1 and 2 the matched Markov chain is used as an index into vectors λ_2 and λ_3. Since it is possible to match more than one Markov chain as shown for input sequence <C1, C2, C3> the single chosen Markov chain is non-deterministic (HTM implementation dependent).

11. Appendix C – Example of Longest Common Subsequence Computation

Consider the following examples to illustrate the use of formulas (1) *LCSm* and (2) *LCSUm*

1. Let *IS* = <a,b,c> and let c*LLS* = <a,b,c,d,x> then
   a. *LCS*(IS,cLLS) = <a,b,c>, IS_1 = <a>, cLLS_1 = <a>, |*IS*| = 3, |c*LLS*| = 5, *ALLL*(IS,cLLS) = |*IS*| = 3, |LCS(IS,cLLS)| = 3
   b. LCS_m(IS,cLLS) = 3/3, LCS_u(IS,cLLS) = <a,b,c>,| LCS_u(IS,cLLS) | = 3, $LCSU_m$ = 3/3
2. Let *IS* = <a,m,c> and let c*LLS* = <a,b,c,d,x> then
   a. *LCS*(IS,cLLS) = <a,c>, IS_1 = <a>, cLLS_1 = <a>, |*IS*| = 3, |c*LLS*| = 5, *ALLL*(IS,cLLS) = max(*IS*,cLLS) = 5, |LCS(IS,cLLS)| = 2
   b. LCS_m(IS,cLLS) = 2/5, LCS_u(IS,cLLS) = <a> and <c>, | LCS_u| = 1, $LCSU_m$ = 1/5
3. Let *IS* = <b,c> and let c*LLS* = <a,b,c,d,x> then
   a. *LCS*(IS,cLLS) = <b,c>, IS_1 = <b>, cLLS_1 = <a>, |*IS*| = 2, |c*LLS*| = 5, *ALLL*(IS,cLLS) = max(*IS*,cLLS) = 5, |LCS(IS,cLLS)| = 2

b. $LCS_m(IS,cLLS) = 2/5$, $LCS_u(IS,cLLS) = <bc>$, $| LCS_u | = 2$, $LCSU_m = 2/5$

The best match is achieved with example (1) since all of the input sequence is matched against the learned sequence. The next best match is example (3) which matches all of the input but not from the beginning of the learned sequence. The worst match is example (2) which matches part of the input, albeit from the beginning of the learned sequence.

12. Appendix D – Computing Path Probability

Assume that each element of sequence LLS has its path probability computed according to equations (11, 12) and probability values stored in table *Learned_LLS_Nodes*, as shown in Fig.17. Then *ComputePathProbability* computes the path probability of IS based on sequence LLS as shown in Fig.17.

```
ComputePathProbability(input, Learned_LLS_Nodes)
// Compute the path probability of the input from the LLS applying appropriate penalties
//for  mismatches as follows:
  Path_prob = 1.0
  For each element "e" of the input (IS) Do
    IF match is found between "e" and learned_LLS_Nodes[i] at the next matched
      consecutive position "i" in sequence LLS
    THEN  // Condition (12.a)
                  -   Path_prob = Path_prob * learned_LLS_Nodes[i].probability
     Else IF "e" does not match any elements in learned_LLS_Nodes from position i  OR "e"
            matches an already matched element of  learned_LLS_Nodes
     THEN // Condition (12.b)
           // Penalize this input element
                  Path_prob = Path_prob * PENALTY
      ELSE IF a match is found between "e" and learned_LLS_Nodes[j] not at the next
           matched consecutive position
       THEN  // Condition (12.c)
             // Elements exist in the learned LLS at a position "j" beyond elements at
             //  position "i" (last matched element) in the learned LLS that are not part of
             //  the input ➔ Penalize them
                  -   Path_prob = Path_prob * learned_LLS_Nodes[i].probability * (j - i )
                      * PENALTY
             EndIF
          EnDo
```

Figure 17 Path Probability Algorithm

For instance, assume IS = $<S_1, S_3, S_4>$ then from Fig.14 LLS = $<S_1,S_3,S_6>$ then

$P(LLS_1) = P(LLS_{start \rightarrow S1}) = 4/16$ since $LLS_1 == IS_1$ from (11)

$P(LLS_2) = P(LL_{S1 \rightarrow S3}) = P(LLS_2 | LLS_1) = 2/4$

since $LLS_2 == IS_2$ from condition (12.a)

$P(LLS_3) =$ penalty $= 0.0001$ since $LLS_3 \neq IS_3$ from condition (12.b)

$P(LLS) = 4/16 \times 2/4 \times 0.0001$

Another example consider IS = <S1,S4> and assume that LLS = <S1,S2,S3,S4> then

$P(LLS_1) = P(LLS_{start \rightarrow S1}) = 4/16$ since $LLS_1 == IS_1$ from (11)
$P(LLS_2) = P(LLS_3) =$ penalty $= (0.0001 \times 2)$
since $IS_2 == LLS_4$ from condition (12.c)
$P(LLS) = 4/16 \times (0.0001 \times 2)$

Another example consider IS = <H1,H4,H5,H2> then from Fig.14 LLS =<H1,H2> then

$P(LLS_1) = P(LLS_{start \rightarrow H1}) = 5/16$ since $LSS_1 == IS_1$ from (11)
$P(LLS_2)$ = penalty = 0.0001 since $IS_2 \neq LLS_2$ from condition (12.b)
$P(LLS_3)$ = penalty = 0.0001 since $IS_3 \neq LLS_2$ from condition (12.b)
$P(LLS_4) = P(LLS_{H1 \rightarrow H2}) = 2/5$ since $IS_4 == LLS_2$ from condition (12.a)
$P(LLS) = 4/16 \times 0.0001 \times 0.0001 \times 2/5$

The first example produces the best match. The second example produces the second best match and the third example the worst match. Compare these results to utilization of longest common sequence and longest common substring calculations and the results are different. When using equations feed forward belief calculations 1 and 2, the second example produces the best match while the first example produces the second best result and the third example produces the worst result. Equations 1 and 2 reward matching all elements of the input sequence *IS*, whereas the probability based computations are more sensitive to any mismatch between input and learned sequence. Another way to look at it, is that longest common based algorithms of similarity match the input more "loosely" than probability based algorithms.

13. Appendix E – Understanding HTM Algorithms

To understand how HTM algorithms work consider the following example where input to the Max Output layer is generated during inference for two HTMs (HTM_A, HTM_B, see Table 19) from the *same observation* (made up of several input sequences totaling 50 web destinations).

Table 19 Feed Forward Beliefs with associated input received at Max Output Layer

HTM_A $\lambda_{OutputK}$	**HTM_A matched input sequence (IS_k) size**	**HTM_B $\lambda_{OutputK}$**	**HTM_B matched input sequence (IS_k)**
0.45	1	0.99	5
0.76	5	0.35	4
0.22	5	0.65	3
0.35	3	0.78	5
0.44	2	0.44	4
0.77	5	0.96	5

0.85	4	0.84	4
0.31	5	0.47	4
0.56	5	0.24	3
0.30	5	0.67	3
0.66	3	0.52	5
0.94	2	0.72	5
0.29	5		

- Average HTM_A = (0.45 + 0.76 + 0.22 + 0.35 + 0.44 + 0.77 + 0.85 + 0.31 + 0.56 + 0.30 + 0.66 + 0.94 + 0.29)/50 = 6.9/50=0.138
- Average HTM_B = (0.99 + 0.35 + 0.65 + 0.78 + 0.44 + 0.96 + 0.84 + 0.47 + 0.24 + 0.67 + 0.52 + 0.72)/50 = 7.63/50= 0.1526
- Weighted Ave HTM_A = [0.45×(1/50)] + [0.76× (5/50)] + [0.22 × (5/50)] + [0.35 × (3/50)] + [0.44 × (2/50)] + [0.77 × (5/50)] + [0.85 × (4/50)] + [0.31 × (5/50)] + [0.56 × (5/50)] + [0.30 × (5/50)] + [0.66 × (3/50)] + [0.94 × (2/50)] + [0.29× (5/50)] = 0.5138
- Weighted Ave HTM_B = [0.99 ×(5/50)] + [0.35 × (4/50)] + [0.65 × (3/50)] + [0.78 × (5/50)] + [0.44 × (4/50)] + [0.96 × (5/50)] + [0.84 × (4/50)] + [0.47 × (4/50)] + [0.24 × (3/50)] + [0.67 × (3/50)] + [0.52 × (5/50)] + [0.72× (5/50)] = 0.6586
- Path Prob HTM_A = 0.45 × 0.76 × 0.22 × 0.35 × 0.44 × 0.77 × 0.85 × 0.31 × 0.56 × 0.30 × 0.66 × 0.94 × 0.29 = 0.000071
- Path Prob HTM_B = 0.99 × 0.35 × 0.65 × 0.78 × 0.44 × 0.96 × 0.84 × 0.47 × 0.24 × 0.67 × 0.52 ×0.72 = 0.00176

For this example, for all HTM algorithms, the observation matches sequences learned by HTM_B better than sequences learned by HTM_A since $HTM_B = \max_{HTMA,B}(\text{Average}()) = \max_{HTMA,B}(\text{Weighted Sum}()) = \max_{HTMA,B}(\text{Path Probability}())$ (see eq. 17).

14. Appendix F - The Design of Synthetic Data

Synthetic train and test data was produced for both the HTM and Markov Chain (MC) based algorithms. The User Attribution solution was verified against synthetic data that mimics user web visits found in real world scenarios as shown in Fig. 18, using the algorithm presented in Fig.19. The User Attribution solution was also verified against MC approaches. These approaches, as opposed to the HTM, do not leverage any timing information. For Markov Chains based approaches tests were performed using the same synthetic data generated by the algorithm in Fig. 19 with the exception that all timing information (time stamp and TS values) was removed so that only sequences of destinations are left to be processed. Training and inference for these alternate approaches took place based on "observations". Each

observation simulated a user web session worth of input and consisted of a predetermined number (50) of web sites visited.

Simulation was performed by using input data that is as representative of real user network traffic as possible. The input to the HTM prototype has the following form: Timestamp<TS, Dest>, where: (1) Generation of the Timestamp input field was accomplished by modeling devices entering (random distribution arrival times) and leaving (random distribution for service times) the network. (2) Generation of the TCP TS value was accomplished by using a 50/50 ratio of TS values started at a fixed value (iphones) and random values (android phones). (3) Generation of destinations (ranked in order of popularity) visited by all users in the simulation follow a power law distribution (Zipf) .

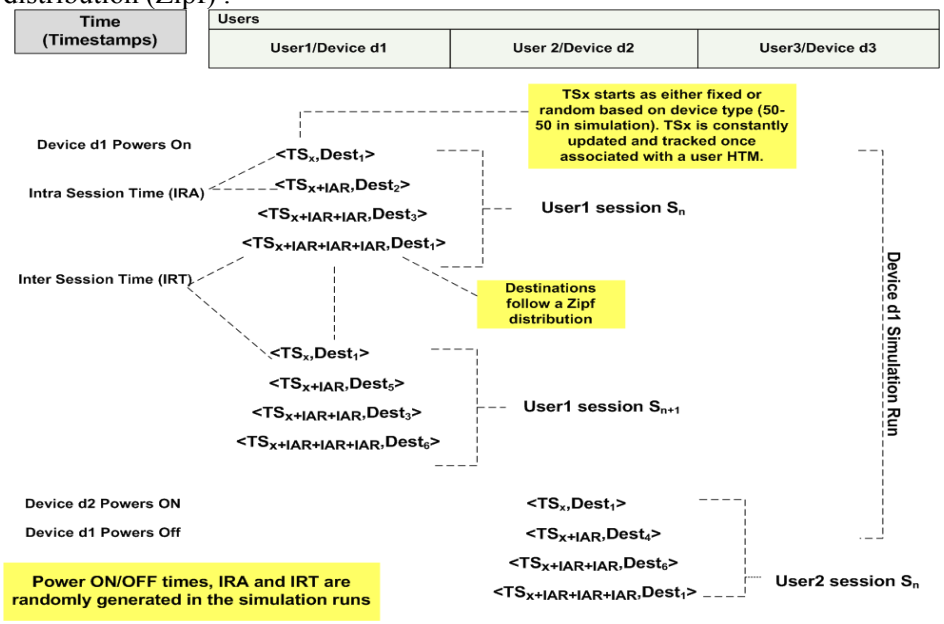


Figure 18 Synthetic Input Data for User Attribution Simulation

Fig. 18 shows the input framework within which the simulation was run. A Java application was developed separate from the HTM, which produced, for each user, the synthetic input data as shown in Fig. 18. The data simulated devices associated with users entering the network at random times and initiating multiple communication sessions until the devices are turned off. Table 20 below shows the various random parameters that were used in the simulation.

Random Simulation Parameters	Statistical Distributions	Boundaries of Distributions	Explanations
Power On Time	Random Uniform	0 – 3 hours	Simulates users powering on their devices and entering the network in the morning hours, between 6:00 AM and 9:00 AM
Intra Session Time (IRA)	Random Uniform	0 – 5 seconds	Time between HTTP requests for a given user within the same user communication session. User communication sessions form clusters of web destinations visited by a user that follow each other close in time.
Inter Session Time (IRT)	Random Uniform	1 – 5 minutes	Time between the end of a user communication session and the beginning of the next user communication session for that same user.
Service Time	Random Uniform	Power Off Time - Power On Time	Amount of time a device once powered on remains on in the network.
Power Off Time	Random Uniform	0 – 21 hours	Simulates time when users power off their devices and exit the network.
Web Destinations	Zipf	1 – 10,000 web destinations	Simulates web destinations ranked in order of importance (1 most visited to 10000 as the least visited) visited by users.
Number of Web Destinations per user session	Random Uniform	1- 10 destination per session	For each user session a user is allowed between 1 to 10 web visits chosen at random.
TCP Timestamp (TS)	Random Uniform	$0 - 2^{32}$	50 % of devices entering the network will have a random starting value while the other 50% will have a fixed starting value of 0.

Table 20 Simulation Parameters

The input generation application creates an input file for each simulated user where the numbers of train and test days are configurable parameters.

The algorithm in Fig. 19 creates 5 simulation days' worth of synthetic train data for user Ux. This simulation code generates synthetic data for training purposes for both HTM and alternate approaches. Each simulation day contains a random number of user sessions bounded by random intersession times. Each user session for the HTM is made up of a random number of input tokens of the form: Timestamp<TS, Dest>. Within a user session, the intra session time randomly spaces occurrences of the input tokens. Destinations are selected based on the Zipf distribution, with the most popular destinations having the highest probability of being selected over less popular destinations. The algorithm used to implement the zipf distribution is based on the zipf algorithm used in [20]. *Next_ZipfRandom* in Fig. 19 returns the next web site

in rank order from 1 to n (with 1 being the most visited and n the least) following a power law distribution. The algorithm generates web sites that are weight proportional to the Riemann zeta function: $\frac{1}{1}^{\theta} + \frac{1}{2}^{\theta} + \ldots + \frac{1}{N}^{\theta}$. In the algorithm in [20], θ(theta) controls the skewness such that θ = 1.0 indicates the highest skew (all nodes have different popularity) and θ = 0 indicates the lowest skew (all nodes are equally popular). For this study, theta was set to 0.96.

```
Ux_Max-Simulation_Days = 5       // Defines max number of Train or Test days

// Create one input file per user Ux in simulation
For Each user Ux in simulation Do
        -     Generate_Input_For_User(Ux, Ux_Max_Simulation_Days)
EnDo

Generate_Input_For_User(Ux, Ux_Max_Simulation_Days)
 TimeStamp                  =          0
 DevicePowerOnTime          =          TimeStamp  + Uniform Random(0, 3Hrs)
 DevicePowerOffTime         =          DevicePowerOnTime     + Uniform Random(0, 21Hrs)
 TS                         =           Generate TCP TimeStamp-TS
 TimeStamp                  =          DevicePowerOnTime
 While (Ux_Max_Simulation_Days  > 0 ) Do
     While (TimeStamp  <  DevicePowerOffTime) Do
          NumberDestinationsPerSessions =  Uniform Random(1,10)
          While (NumberDestinationsPerSessions  >  0 AND TimeStamp  <
                  DevicePowerOffTime) Do
                Dest       =  Next_ZipfRandom (1000,theta)  // Get the next destination
                Output TimeStamp<TS,Dest> to Ux file name
                NumberDestinationsPerSessions = NumberDestinationsPerSessions – 1
                IF (NumberDestinationsPerSessions > 0) THEN
                          IntraSessionTime-IRA =         UniformRandom(0,5secs)
                          TimeStamp            =         IntraSessionTime-IRA
                           TS                  =           TS + IntraSessionTime
                EndIF
           EnDO
           InterSessionTime-IRT       =          UniformRandom(1,5mins)
           TimeStamp                  =          InterSessionTime-IRT
           TS                         =          TS  + InterSessionTime
     EnDO
     Ux_Max_Simulation_Days  =      Ux_Max_Simulation_Days  – 1
 EnDO
```

Figure 19 High level algorithm to generate synthetic random train input for a single user

Test data had to be created using a different approach since it had to be similar to the train data but also maintain a certain level of independence from train data. Three methods were used for generation of synthetic data for the test phase of experiments. All three algorithms (Random Walk, Walk Only and Concept Drift) walk a first order Markov chain of learned destinations which

were generated during the training phase of the synthetic data generation process.

In the "Random Walk" the next destination V_j, for transitions of the form $V_i \rightarrow V_j$, is chosen randomly in proportion to the in-degree of the node V_j. That is, in proportion to the access frequencies of the neighbors ($V_{j1}, \ldots V_{jn}$) of the current node (V_i). If no such neighbor V_j exists then the walk proceeds with a new node V_i with at least one neighbor, selected from the learned destinations based on a zipf distribution. Selection of the next destination V_j is based on the work of Price (1976) [39] who proposed a model of networks formation that gives rise to power-law degree distributions. Price was interested in the power law distribution of citation networks. Specifically, his model showed that a newly appearing paper cites previous ones chosen at random with a probability proportional to the number of citations that those previous papers already have. This property is critical in creating a relationship between train data generated for a given user with test data for that same user. While a relationship must exist between the train and test data sets it must also maintain a certain level of independence between the two sets which is provided by the randomness of the selection of already visited nodes. While the Price model has been applied to simulation of networks traversed by many users, in this study, this model is adjusted to simulate web visits by a single user. As a result the emphasis was not placed exclusively on in-degree or out-degree of network nodes but instead on the frequencies of edges emanating from or terminating to nodes representing web visits to web sites. The algorithm follows with connectivity probability

$$\mathbf{1} - \frac{Oi}{Oi+Ci} > \mathbf{r} \ (\mathbf{0} \leq \mathbf{r} \leq \mathbf{1})$$

a learned path proportional to the frequency of the in-degree of web sites along the path. Otherwise it starts a new path. In terms of notation, **r** is a random number that follows a uniform distribution**,** $\mathbf{C_i}$ represents the sum of traversal frequencies of all edges emanating from V_i ($V_i \rightarrow V_{j1-n}$) and $\mathbf{O_i}$ is the out degree of $\mathbf{V_i}$. As would happen in real life the algorithm favors learned path patterns, but does also produce variations that simulate "concept drift". In "Walk Only" -, the algorithm selects V_j randomly in proportion to access frequencies of all of Vi's neighbors as long as $\mathbf{V_i}$ has at least one neighbor. Note that this algorithm minimizes any concept drift since it always follows a learned path as long as one exists, as opposed to the Random Walk algorithm that is constrained by the connectivity probability and the random value of **r**. In "Context Drift", the algorithm selects V_j using the Walk Only algorithm except for 20% of the V_j destinations that are selected as new ones outside of the learned train set. In addition, 10% of the $V_i \rightarrow V_j$ transitions selected during the walk are new (not existing in the train set).

14.1. Evaluating similarity between Synthetic and Real Network data

In order to determine how similar train and test data sets are to each other, similarity statistics were computed against real cellular network data collected for an equivalent number of users. Observation similarity statistics between train and test data sets were generated based on all observations processed leveraging the work of Kumar and Raju [31].

(OSS)Observation Sequence Similarity = (21)

$$\sum_{i=1}^{|TO|} \frac{|LCS(TOi,\ TrO)|}{|TOi|} \Big/ |TO|$$

(OSuS)Observation Substring Similarity = (22)

$$\sum_{i=1}^{|TO|} \frac{|LCSu(TOi,\ TrO)|}{|TOi|} \Big/ |TO|$$

(OSeS)Observation Set Similarity = (23)

$$\sum_{i=1}^{|TO|} \frac{|TOi\ \cap TrO|}{|TOi|} \Big/ |TO|$$

Overall Similarity = (24)

(.33)×OSS + (.33)×OSuS + (.33)×OSeS

TO and *TrO* are the sets of all test and train observations respectively. TO_i is a specific test observation from the test set *TO*. $LCS(TOi,\ TrO)$ is the length of the longest common subsequence match between a specific test observation and all train observations, whereas $LCSu(TOi,\ TrO)$ is the length of the longest common substring match between a specific test observation and all train observations. As can be seen from Fig. 20 overall similarity between train and test synthetic data sets is 50%, with set similarity (observations in train and test data sets containing the same destinations but not in the same order) being as high as 83%. Sequence and substring similarity measure how alike sequences of destinations are between train and test data sets. The real network data measurements (line in red in Fig. 20) for an equivalent data set (5 users, 5 train days, and 1 test days) collected from a real network show that train and test data sets are more similar to each other than similar data sets derived from synthetic data. These results provide support for the belief that synthetic data

represents a good benchmark for baselining the experiments conducted in this study.

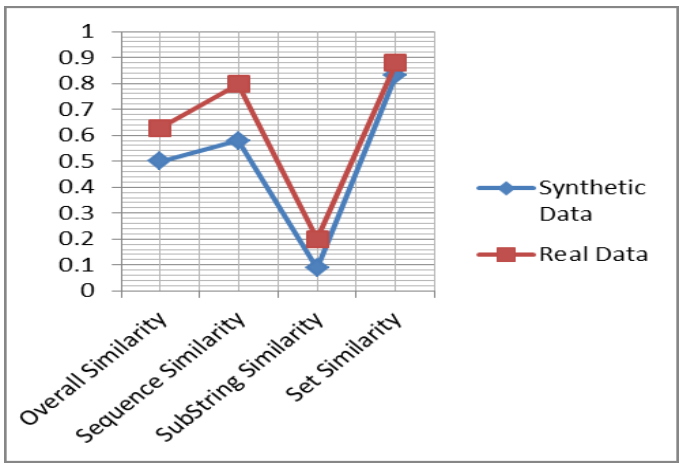


Figure 20 Similarities between Train and Test Data Sets

15. Appendix G – Why use HTMs when dealing with Complex Networks?

HTMs are unique in stressing the temporal aspect of perception and implementing memory for sequences of patterns that facilitate anticipation. Each level in the hierarchy is trained separately to memorize spatial-temporal objects (patterns) and is able to recognize objects in a bottom-up/top-down process [18]. The HTM hierarchy also enables efficient representation of relationships among many inputs by leveraging reuse of lower level inputs in order to represent higher level concepts at higher levels of the hierarchy. HTMs allow sequence learning (concatenation of spatial and then temporal learning), which provides the ability to make predictions and can be applied to disambiguate input. Only few methods exist that combine spatial and temporal learning in a tight way (e.g. recurrent neural networks can do this a well) [22].

Of specific interest to this study is the evaluation of the distribution of visitors to web sites. Adamic and Huberman [1] studied the distribution of users among web sites by examining usage logs from America Online covering 120,000 sites. They discovered that the distribution of visitors per site follows a universal power law similar to that found by Pareto in income distributions. They reasoned that a small number of sites control the traffic of the web population, a result typical of winner-take-all markets. The authors agree that

the World Wide Web gives rise to an asymptotic self-similar structure in which there is no natural scale and the number of users per site is indeed distributed according to a power law. In another study, Adamic and Huberman [2] find inconsistencies in the conclusions of a study by Barabasi and Albert [5] which states that because of preferential treatment, a vertex that acquires more connections than another will increase its connectivity at a higher rate so that the connectivity between nodes increases in line with the growth of the network. This leads to older vertices increasing their connectivity at the expense of younger and leading to the well known "rich-get-richer" phenomenon for highly connected vertices. Adamic and Huberman studied web crawls of 260,000 sites and concluded that all sites are not created equal since no correlation exists between the age of a site and its number of links. They explain that the rate of acquisition of new links varies from site to site and is probably proportional to the number of links the site already has, because the more links the site already has, the more visible it becomes and the more links it will get.

While there has been agreement in the research community that communication traffic has self-similar characteristics, until recently it was believed that complex networks are not invariant or self-similar under large scale transformations. This belief is rooted in the small world property of these networks which would seem to imply that the number of nodes increases exponentially with the diameter of the network rather than following the power law relation expected for self-similar structures. Song, Havlin and Maske [44] analyzed real complex networks, like the web, utilizing a box counting method as a scale invariant renormalization procedure and concluded that, on the contrary, these networks consist of self-repeating patterns on all length scales that suggest they share common self-organizing properties.

What are the implications of addressing the user attribution problem in the context of complex networks? The self-similar, small world and clustering properties together with the preferential attachment characteristic of complex networks supports the notion that users tend to visit a limited number of mostly popular sites with increasing frequencies. How can the approach implemented in this study leverage unique and personal patterns to differentiate among users if different users visit mostly the same sites and this research proposes to use web site visits as a way to uniquely recognize users?

This study has leveraged the power law properties that characterize web traffic of users who visit different web sites. Specifically, the implications of the power law distribution support the notion that while it is true that few web sites get visited very often by all users, many web sites, in the long tail portion of the power law distribution, get visited less often by a variety of users as well. By recording communication patterns of past activity for each user it becomes possible to identify unique and differentiating elements that will enable isolation among users. More specifically, the hypothesis in this study has been that the long tail properties of the distribution of user visits to web

sites together with the time order of such visits create conditions for unique differentiation among user patterns that allows to adequately address the user attribution problem.

HTMs have been successfully used in classification problems in a variety of applications such as recognition of USPS handwritten digits [8], speech recognition [17], and prediction of user choices on mobile phones [34]. HTMs have also been used in the area of web analytics which represents an important use case for this study. In a talk given for the association of computing machinery (ACM) in 2009, Subutal Ahmad, vice president of engineering at Numenta, described results of experiments using Numenta's HTMs to *predict* user web click behavior for topics and pages of interest to the user. In these experiments web content was partitioned into 177 different topics. In their experiments random prediction reported 0.56% accuracy. By training the HTM with 100,000 user sequences (web pages) and using no temporal context (0^{th} order prediction based on recorded popularity of topics and web pages) the accuracy reported was 23%, which matches what most web sites can do today. By including in the analysis transition probabilities from a given web page to another in the form of 1^{st} order prediction, predictive accuracy increased to 28%. By further leveraging use of variable order prediction, accuracy levels jumped to 45%. Variable order prediction, used in this study, allows prediction to fully leverage the dynamic "context" (different length sequences) of web pages visited by a user.

16. Appendix H – Tables Describing Parameters for Experiments

Table 21 . EXPERIMENT TYPE E1, USER ATTRIBUTION NO CONCEPT DRIFT

Number of Web Destinations	**Number of Users**				
	5	**20**	**50**	**100**	**500**
1000	5/1, 5/2, 3Obs	5/1, 5/2, 3Obs	5/1, 5/2, 3Obs	5/1, 3Obs	3Obs
5000	5/1, 3Obs	5/1, 3Obs	5/1, 3Obs	5/1,3Obs	3Obs
10,000	5/1, 3Obs	5/1, 3Obs	5/1, 3Obs	5/1,3Obs	3Obs

Table 22 . EXPERIMENT TYPE E2, USER ATTRIBUTION WITH CONCEPT DRIFT

Number of Users	**Number Train Days/Number of Test Days**			
	5/2	**10/3**	**15/4**	**20/5**
5	Walk Only,	Walk Only	Walk Only	Walk Only
5	Random Walk	Random Walk	Random Walk	Random Walk
5	Walk Only 20%	Walk Only 20%	Walk Only 20%	Walk Only 20%

Number of Users	Number Train Days/Number of Test Days			
	5/2	**10/3**	**15/4**	**20/5**
	and10%	and10%	and10%	and10%

Table 23. EXPERIMENT TYPE E3, E10 USER ATTRIBUTION UNDER DOS ATTACK

Number of Users/Number infected users	DOS Attack Parameters		
	Number Destinations/Unit of Time, Repeats Every	**Number Destinations/Unit of Time, Repeats Every**	**Number Destinations/Unit of Time, Repeats Every**
10/4	5/5ms, 5ms	10/10ms, 5ms	20/20ms, 5ms

Table 24 . EXPERIMENT TYPE E4, E11 USER ATTRIBUTION UNDER PHISH ATTACK

Number of Users/Number infected users	Phish Attack Parameters		
	Number Destinations/Unit of Time, Repeats Every	**Number Destinations/Unit of Time, Repeats Every**	**Number Destinations/Unit of Time, Repeats Every**
10/4	1/1ms, (1min-1hour)	3/3ms, (1min-1hour)	5/5ms, (1min-1hour)

Table 25. EXPERIMENT TYPES E7,E8, E9, USER ATTRIBUTION

Number of Users	Number Train Days/Number of Test Days		
5	**5/1**	**5/2**	**10/3**
10	**5/1**	**5/2**	**10/3**

17. Appendix I - Glossary of HTM Abbreviations and Symbols

The table below provides an explanation for many of the abbreviations, terms and symbols used in HTM computations and algorithms found in the "The Approach" section.

Table 26. TERMS USED IN HTM CALCULATIONS AND ALGORITHMS

Terms	Description
ALLLS	*Adjusted Length LLS c*omputes the appropriate proportion of an input string *IS* matched against the longest common subsequence *LCS* or longest common substring *LCSu* of a candidate learned longest common subsequence *cLLS* that needs to be accounted for during similarity calculations. Specifically, *ALLLS* returns the length of the input sequence *IS* matched against subsequence cLLS. This length, based on the *ALLLS_Cond* condition, is equal to the size of the input sequence when *IS* matches completely from the beginning sequence *cLLS*, otherwise the length returned is the size of the longer sequence between the *IS* and *cLLS*. Experiments conducted during this study have shown that matching substrings from the beginning of a best matching *cLLS* produces better recall accuracy results than matching substrings sequences in the middle of *cLLS*.
ALLLS_Cond	*Adjusted Length LLS condition* is true if the size of the input sequence *IS* matches completely from the beginning sequence *cLLS*.
cLLS	*cLLS* refers to a candidate longest learned sequence *LLS*, one out possibly many that matches *IS*, out of all Markov chains in a given HTM layer.
DM	*Degree of Membership* finds the best (longest) match computed based on sequence similarity (SS) between a single input sequence *IS* and all *cLSS* sequences in a given HTM layer. See equation (8)
$\boldsymbol{FFB_{gi}}$	*Feed forward belief* measures the degree of membership (*DM*) of a given input sequence at layer L_{x-1} of the HTM computed against the most persistently visited *LLS* belonging to Temporal group g_i at layer $L_{x-1.}$ Each computed LLS can only belong to a unique Markov Chain within an HTM layer. See equation (9).
FFB_PP	*Path Probability of LLS* is the path probability algorithm which computes the path probability of matching the longest learned sequence (*LLS*) and then for each *LLS* mismatch against *IS* a penalty is computed. The algorithm ensures that path probability $P(LLS_k \mid LLS_j)$ (where LLS_k follows directly LLS_j) is computed only if LLS_k and LLS_j are both matched in *IS* otherwise penalties are computed (as shown in condition 12.c of the path probability algorithm shown in Fig. 17). See equation (10).
Fq	*Frequency of visits* is used to compute the persistence (*PS*) of a given cLLS sequence.
gi	A temporal group *gi*, also known as "coincidence", is the Markov chain which holds the Longest Learned Sequence (*LLS*)
IAL	*Input Activation Level* represents the strength of the match of the input sequence against the longest learned sequence (*LLS*) from the HTM layer below L_{n-1} and is used to normalize feed forward belief calculations at layer L_n. Examples of how feed forward beliefs propagate through HTM layers are shown in Appendix B. See equation (16)

$\boldsymbol{\lambda_{Lx}<FFB_{gi}, FFB_{gi+1,}>}$	A vector of feed forward beliefs at HTM layers 1, 2, 3. The topmost HTM layer is also known as "output" layer. See equation (15) for λ_{L2} and equation (14) for $\lambda_{L3.}$
$\boldsymbol{\lambda_{Output<}FFB_{gi}, FFB_{gi+1,}>}$	A vector of feed forward beliefs at the output HTM layer. See equation (13) for $\lambda_{Loutput}$
LCS	*Longest Common Subsequence* is computed as the longest sequence of inputs (web destinations or temporal groups) that appears left to right but not necessarily in a contiguous block in both input sequence (*IS*) and the matched *cLLS*.
$\boldsymbol{LCS_m}$	*Longest Common Subsequence Measure* computes the portion of the *cLLS* matched as the longest common subsequence against the input sequence *IS*. See equation (1).
LCSu	*Longest Common Substring* is computed as the longest sequence of inputs (web destinations or temporal groups) that appears left to right in a contiguous block in both input sequence (*IS*) and matched *cLLS*.
$\boldsymbol{LCSU_m}$	*Longest Common Substring Measure* computes the portion of the *cLLS* matched as the longest common substring against the input sequence *IS*. See equation (2).
LLS	*Longest Learned Sequence* is the single longest sequence of learned inputs (web destinations or temporal groups) in a Markov chain within an HTM layer that best matches the input sequence. The *LLS* is discovered by computing the longest common subsequence and substring of learned sequences in the HTM (eq. 1,2).
MAX_FFB_HTMs	*Max Feed Forward Belief for HTMs* uses the $max_{HTM1..M}$ function to compute the highest valued feed forward belief (based on one of 7 HTM algorithms) for a given observation worth of input for a given HTM/user (see equation 17). To review examples of calculations of feed forward beliefs propagating towards upper layers of the HTM during HTM training see Appendix A and during inference see Appendix B.
Ps	*Persistence* measures how often learned sequences in Markov chains are visited (see equation 4). For instance, consider the Markov chains at layer 1 of the HTM in Fig. 14 from Appendix B, which used the spatial pooler algorithm to combine groups of web destinations into sequences based on terminating conditions $TC_{1\text{-}3.}$ These Markov chains were created during the training phase using sequences of web sites as shown in Table 10 from Appendix A. All nodes in Fig. 14 represent unique web destinations, with the exception of cloned nodes which are represented using prime symbols as in S3' and S3''. Assume that cLLS = $<S_1, S_2, S_3, S_4>$ then Fq(cLLS) = 1, NLS = 16 and Ps(cLLS) = 1/16. As another example, assume cLLS = $<S_1,S_3,S_6>$ from Fig. 14, then Fq(cLLS) = 2, NLS = 16 and Ps(cLLS) = 2/16.
$\boldsymbol{S_{gi..i+N}(g_i, g_{i+1}, g_{i+2,...})}$	Sequence of temporal groups received from a lower HTM layer which represents an input sequence learned and inferred within variable order Markov chains at HTM layers 2 and 3
$\boldsymbol{S_{\lambda Output k..k+N}(\lambda Output_{k+N})}$	A sequence of feed forward beliefs received at Max Output layer from a given HTM_i for a single observation worth of data and represents an input sequence (*IS*) received at the Max

	Output layer.
Sp	*Sequence persistence* measures the persistence (frequency) of visits (*Ps*) to a given *cLLS* normalized using persistence weight W_P. See equation (7).
SS	*Sequence Similarity* measures the similarity of an input sequence (*IS*) and *cLLS* based on the aggregate of longest common subsequence (LCS_m) and substring measures ($LCSU_m$) normalized with subsequence and substring weights (*Ws*,*Wu*). See equation (6).
Timestamp<TS, Dest>	Represents HTM input at layer 1. Timestamp is the time in milliseconds when the input was received by the HTM. TCP Timestamp (*TS*) is the time in milliseconds when a TCP packet was sent. Destination (*Dest*) is the destination address (represented within the HTM as a number) of the HTTP request received by the HTM.
Ws	Similarity weights represent the relative importance attributed to different components of the sequence similarity calculation. *Ws* represents the weight given to the longest common subsequence portion of the sequence similarity calculation. *Wu* represents the weight given to the longest common substring portion of the sequence similarity calculation. *Wp* represents the weight given to the sequence persistence calculation. The similarity weights values are based on results from experiments which have shown that sequence persistence (*SP*) which is simply based on frequency of occurrences of learned sequences, while necessary to improve overall accuracy results, has the least impact on overall recall accuracy in contrast to the sequence similarity calculation (*SS*) which instead is based on recognition of the timed order of web destinations within learned sequences.