2014

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

Claudio Taglienti
Nova Southeastern University, ctaglienti@comcast.net

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

by

Claudio Taglienti

A dissertation submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

in

Computer Information Systems

Graduate School of Computer and Information Sciences

Nova Southeastern University

2013

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

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

by

Claudio Taglienti

October 2013

In the context of 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 is analogous to trying to recognize a nameless face in a crowd. This problem worsens as users move across many mobile networks (complex networks) owned and operated by different providers. The traditional approach of using the source IP address, which indicates where a packet comes from, 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 were limited to a small number of users (28 and 100 users). This was due to the inability of solutions to link up multiple user sessions together when they rely exclusively on the 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. Each layer of an HTM contains variable order Markov chains of connected nodes which represent clusters of web sites visited in time order by the user (user sessions). HTM layers enable inference "generalization" by linking Markov chains within and across layers and thus allow matching longer sequences of visited web sites (multiple user sessions). This approach enables linking multiple user sessions together without the need for a tracking identifier such as the source IP address.

Results are promising. HTMs can provide high levels of 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. This research confirmed that the presence of long tail web sites (rarely visited) among many repeated destinations can create

unique differentiation. What was not anticipated prior to this research was the very high degree of repetitiveness of some web destinations found in real network data.

Acknowledgements

There are many who have made this journey possible. I am thankful to our Lord for always supporting me and my family during this challenging process. I could have not completed this difficult journey without the unconditional love and care of my wife Dianne and my children Luke and Cristina.

At the heart of this work is the supervision of my advisor, Dr. Cannady who was instrumental in guiding me during the choice of the specific approach and throughout this process. Dr. Cannady's belief in me has given me the confidence to tackle the many trials that have emerged during the dissertation. I am grateful to Dr. Mukherjee who has challenged the original approach requesting an early test to verify the validity of the proposed solution. By successfully completing those tests both the dissertation committee and I gained confidence in the proposed approach. Special thanks to Dr. Li for helping me improve the quality of the dissertation by encouraging me to find the right words to adequately express concepts and expand on key topics of this work.

I would also like to acknowledge Narothum Saxena and Mike Irizarry for agreeing to sponsor this degree and for their continued support for my efforts over the years. I am grateful to my coworkers Slava Lemberg and Sebastian Thalanany for never getting tired of listening to my ideas. Finally, I want to thank Sarin Virani and Patrick Chen for their assistance in collecting communication traffic data from the cellular network.

I made several friends while attending Nova Southeastern University, but one in particular stands out Daffyd E. MacSteaphan (formerly David O. Schankin) as the one I will miss the most.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

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. Clark and Landau (2010) acknowledge the need for stronger forms of personal identification that can be observed in the network and define 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 but solutions are strongly coupled to 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. The re-identification approach has been successfully applied to email-alias detection, author attribution 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 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.

Problem Statement

This study 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 device or 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 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.

Recognition of the source of communication is especially important in the security field where it is necessary to allow users/devices classified as malicious to be blocked or to have their communication rate limited when they attempt to re-enter the network irrespective of their new credentials or of the new assigned source IP address. For instance, an intrusion detection and prevention system can detect and block an active user session that has been classified as malicious. However, when the user re-enters the network, especially in mobile networks, that

user looks like a brand new user and the intrusion detection and prevention system needs to quickly rediscover that user as being malicious in order to be able to stop him. An optimal solution should recognize the user, by leveraging historical observations of communication activity, before any malicious behavior is ever re-started.

The problem of correctly recognizing users by just leveraging user past communication activity (the focus of this study) is generic and not peculiar to the type of network (wireline or wireless) being used. The user attribution problem, when a user accesses a wireline type network (cable, dsl), can be easily addressed by leveraging the source IP address assigned to this user since it changes very seldom. The user attribution problem becomes difficult to solve when users either move across networks or use networks that hide or modify the source IP address as is the case in certain type of mobile networks.

In order to better appreciate the severity of this problem, consider a malicious user or a compromised device 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/device 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 (Casado & Freedman, 2007):

- 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 consequence of the difficulty of utilizing the source IP address as a way to identify the origin of communication is particularly felt in networks where users move as is the case in mobile networks. In these networks no effective method exists to identify a mobile device location by utilizing only the device communication traffic information. Specifically, the properties of mobile devices make IP-based user and device identification, and IP-based geo-location (the problem of locating an internet host by using only its IP address) identification almost impossible to use in order to find a device and its physical location (Balakrishnan, Mohomed, & Ramasubramanian, 2009).

Balakrishnan et al. (2009) are not sure as to the cause of the problem, yet the answer can be found in the way these networks are designed. Mobile networks that serve a large number of subscribers leverage wireless gateways to support mobility. These wireless gateways, allocate IP addresses to devices that have been authenticated and authorized to access the network, and can support mobility for hundreds of thousands of devices across large geographical areas. These wireless gateways own large pools of IP addresses that are dynamically assigned to devices

potentially dispersed across many states within the continental USA. As a result, even in the best of circumstances, where the IP address of a device is traced back to this gateway, it is not possible to identify the device relying solely on the IP address alone since a device in San Francisco could be assigned an IP address (for the duration of that data session) by a wireless gateway deployed in Chicago. The source IP address can also change when users move across wireless gateways (like WIFI access points) that control access to wireless networks that do not support mobility as is the case when a student moves across a university campus.

Traditional security methods that utilize "IP trace back" techniques fail to identify the source of communication originating from devices that operate in complex networks due to the deployment of these large wireless gateways that control the source of communication (source IP addresses) for millions of devices. In addition, identification of the source of communication is complicated by the dynamic assignment of source IP addresses to devices by these wireless 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 device since IP addresses allocated by wireless 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 (Egevang & Francis, 1994).

Dissertation Goal

A new approach is needed that can recognize a user in the network solely based on prior observed communication behavior independently of the IP address assigned to the source or the complex networks that are traversed by the communication traffic. The goal of this study has

been to address the user attribution problem, in its generic form, independent of the type of network.

Research Questions

Specifically, the following questions have been addressed in this study:

1. 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 in terms of better precision, better recall, fewer false positives and fewer false negatives as when using comparable classification approaches?
2. Is the solution scalable? That is, can the solution maintain the same level of performance in terms of accuracy, as the communication population (number of sources and number of destinations contacted by these sources) increases?

The consequences of dynamically changing or of hidden source of communication (IP addresses) has high relevance to the area of network security. This study has addressed the problem of user identification and attribution by forgoing use of tracking techniques like cookies, logins, source IPs and instead it has only leveraged the historical unique communication traffic characteristics of different users. User attribution through behavioral patterns is a new area of research with applications in fields as diverse as marketing and security. Yang (2010) proposes behavior based identification in the context of creation of user profiles built on web usage patterns. Her paper is among the first to study user behavior patterns in web usage data for the purpose of user recognition. She acknowledges that there is currently no research on building

user behavioral profiles from web usage data for the purpose of user recognition and she admits that very limited research is being done on analyzing web users' behavior for user recognition.

The contribution of this proposed work is significant and original in that solutions to address this very important problem are few and limited. The problem addressed is real, difficult to solve, and is important since solutions to this problem can easily be generalized to be applicable in several different domains.

Relevance & Significance

This study addresses the important problem of real time user attribution. A relevant property of user attribution 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 the network (possibly an operator network) and monitors all HTTP traffic. As traffic passes through the user attribution solution (UAS), the solution learns to recognize users (User 1, User 2, User 3, …., User N) based on the user past communication behavior. After the learning stage, the UAS can recognize any user (inference stage) that it has seen before as they re-enter the network possibly using a new source IP address and new authorization credentials without knowing their specific identity (User1 is Joe Smith). A key measure of how successfully the UAS performs is based on the accuracy of user recognition.

The “user attribution problem” is generic and not necessarily tied to attack scenarios, but it can be used to recognize and stop malicious users. Consider a second real-world scenario where the UAS is coupled to an intrusion detection and prevention system (IDPS) so that 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) provided by the UAS. The UAS

provides user labels to the IDPS after it has completed the learning stage and it has entered the inference stage. Assume the IDPS (out of scope for this study) detects anomalous behavior with User4 and blocks all HTTP traffic associated with this user. User4, unable to access the internet shuts down his device and decides to re-enter the network next day. User4 has a brand new IP address assigned and initially 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. The key to the user attribution problem is the ability to learn past user communication behavior so that a user can be recognized.

Recognizing a user re-entering the network is a problem made difficult by the dynamic nature of communication source identifiers, such as source IP addresses, and the limited scope of session identifiers such as cookies that are valid only within a specific web-site visit and not across multiple web-sites visits. These challenges render ineffective the use of current techniques for user identification and user attribution within and across networks. The inability to provide timely and correct classification of the source of communication has serious consequences in the area of security prevention where it is critical to correctly recognize users that have been labeled as malicious so that they can be quickly stopped from re-entering the network.

Online services often use IP addresses as client identifiers when enforcing access-control decisions. Casado and Freedman (2007) in their work acknowledge that the academic community has typically eschewed this approach due to the effect that NATs, proxies, and dynamic addressing have on a server's ability to identify individual clients. Casado et al., recognize the drawbacks of using IP-based identification in the face of NATs, proxies, and dynamic IP renumbering since blacklisting large proxies or NATs can result in legitimate clients losing access to desired services, while whitelisting can give access to unwanted clients. The

authors admit that the actual extent of the problem has remained largely a mystery. They report that part of the challenge in uncovering the impact of edge opacity is a lack of practical techniques and deployments to "peer through the shroud" of middleboxes in the wild. In their study, the authors analyzed data mostly from wireline residential ISPs from nearly 7 million clients across 214 countries and their results show that while 74% of clients are behind NATs or proxies, most NATs are small and follow an exponential distribution. Dynamic renumbering from DHCP generally happens on the order of days with fewer than 2% of the clients that visited their servers over a week's period using more than two IP addresses. Proxies, on the other hand, service client populations that may be both larger and geographically diverse. The authors conclude that poor access control policies for proxies can have greater negative implications than for NATs.

These results seem to indicate that the problem of dynamic changes to source IP is not too serious in a typical residence for NATs since only few devices sit behind them and that dynamic IP addresses assigned to devices persist for relative long time periods. Only web proxies are identified as a problem since these network elements serve a large number of clients. Unfortunately, the impact of NATs, proxies and dynamic IP address changes becomes very serious when considering the impact of mobility and the use of mobile devices in the context of operator networks. On February 3rd 2011, IANA handed out the last blocks of addresses to the Regional Internet Registries (RIRs). As a result there are no more IPv4-addresses at the IANA level, and the depletion is nearly final. The slow depletion of available public IPv4 addresses has encouraged deployment of large scale carrier grade NAT devices in operator networks (Donley, Howard, Kuarsingh, Chandrasekaran, & Ganti, 2010; Jiang, Guo, & Carpenter, 2011). These network devices serve millions of subscribers for both wireless and wireline operators utilizing a

small number of public IP addresses producing the same negative impact that Casado and Freedman reported for web proxies.

The dynamic nature of source IP addresses assigned to users is likely to persist for years to come. IPv6 promises to address the public IPv4 address shortage and eliminate middleboxes. However, source IP addresses will continue to change dynamically even with IPv6, since IP addresses are always assigned by the network (the wireless gateway or access point) to the devices that attach to it (Nakhjiri & Nakhjiri, 2005). This is because the network must advertise reachability of any IP address assigned to a given device in order to ensure that traffic originating from a given device will always return to the originating network and be routed back to that device (Halabi, 2001).

The problem of dynamically changing source IP addresses can occur across a variety of wireless networks. Consider a laptop connecting to a home wireless router, after traveling to the airport it connects to a hot spot provider and then after landing at a new location, this same laptop connects to a business network and accesses the internet. On the way back to the airport, in a shared cab ride the owner of the laptop connects to the internet thanks to the WIFI hotspot functionality provided by an Android phone owned by another rider. The owner of the Android device gets dropped off at his house where his Android phone attaches to the home WIFI wireless router to access the internet. In this scenario, each time a device attaches to a new network that device will be assigned a new IP address. Mobility is pervasive and as users start to move across complex networks, this problem will only worsen.

The consequence of being unable to recognize the source of communication has profound impacts on security and particularly the area of intrusion prevention since attempts to block previously detected malicious devices can back fire. Xie, Yu, and Abadi,(2009) describe the

challenges faced by intrusion detection and prevention devices as they attempt to target containment to offending IP addresses. Detected anomalies cannot be blocked based on the offending source IP address and when an attacker changes its IP address, legitimate activities that subsequently use the old IP address will be misclassified as bad, while malicious activities from the new IP address will slip through. The use of proxies and NAT devices also imply that blacklisting can result in denial of service to many legitimate clients that share IP addresses with attackers.

In the context of mitigating intrusions, one major challenge is the ability to timely and correctly identifying sources of an attack (Peng, Leckie, & Ramamohanarao, 2007). Proper identification is needed to provide a targeted response to the intrusion that can stop the attackers without affecting the rest of the system. This is a difficult problem to address since during an intrusion, the reported source of the attack (namely the source IP address) could be spoofed and made appear to originate from many different valid subnets. It is important to realize that accuracy of identification of a source of an attack is inversely proportional to the scope of impact to the system. Identifying an individual machine or user as the source of an attack allows for a targeted response to stop this attack without affecting any other machine or user in the system. However, if the source of the attack cannot be confined to an individual machine/s or user/s then the scope of the response could expand to the location/s where the intrusion originated from. Mitigation in this case, would entail finding locations in a network where the intrusions originate from and stopping all traffic originating from these locations. It is not difficult to see that with mobility, intrusion mitigation becomes an increasingly complex problem to address.

Accurate and effective response to a detected intrusion is a critical part of a mitigation system. Consider the set of possible actions available to an intrusion response system as

described in (Carver, 2002): Terminate User Session, Block IP Address, Warn the intruder, Trace the connection, Force Additional Authentication, Restrict user activity.

How can any of these activities be carried out when the source of the attack cannot be recognized? A major limitation of current active intrusion response systems is that they do not take into account the collateral damage to legitimate users. These systems can launch immediate responses to attacks, however, predictable responses as described above, can be easily compromised by an adversary by forcing the triggering of massive denial of service attacks against the network by utilizing a large number of spoofed IP addresses (Ghorbani, Lu, & Tavallaee, 2010).

The importance of a reliable source IP address is critical not only in intrusion protection but also in detecting port scans (set of connection attempts from a single source to a set of targets during some time interval). Gates (2009) in her work on port scan detection, acknowledges this important point by indicating that IP addresses are commonly used to represent a source, and as it is often difficult to determine if two different IP addresses represent the same source, in her paper she used IP addresses to indicate the source, recognizing that she may have done so erroneously in some case.

Definition of Terms

The following key terms are used and are uniquely relevant to this study:

- Accuracy – Identification precision measured using one of the following statistical measures: Recall, Precision, False Positives, False Negatives

- Accuracy Scalability – It measures the ability of the solution to maintain consistent (possibly high) levels of accuracy when the number of users and/or the number of web destinations visited increases.
- Attribution – Generally means assigning a cause to an action. For this study it refers to identifying the agent responsible for the action (Clark & Landau, 2010)
- Attribution Problem – David Clark described the attribution problem in terms of the following question: "Why don't packets have license plates?"
- User-Attribution Problem – The problem of identifying a user based solely on network traces of past communication activity without leveraging any authentication or tracking techniques like cookies, logins, source IPs.
- Complex System – Self-organizing systems, which at the end of their evolution show an emergent architecture with unexpected properties and regularities (Barrat, Barthelemy, & Vespignani, 2008).
- Complex Network – A complex system structured as a graph that has non-trivial topological characteristics that do not match previously studied random graphs and has the following properties: self organized dynamic evolution, emergence of the "small-world" concept, structurally scale free and showing a power law degree distribution (see section "Complex Networks" for more details).
- Mobile Networks – Networks that share the properties of complex networks. They are owned and operated by different providers who enforce different administrative policies and can be accessed by mobile devices.

- Run-Time Performance Scalability – It measures how well the solution manages computing resources to accomplish its task in terms of CPU and memory utilization
- User Classification - The process of matching records of past behaviors that belong to the same individual, sometimes when the individual is acting anonymously

Summary

Solutions to the user attribution problem, in the context of complex networks, are relevant in several technology areas such as web mining and security. Because there is so little research in this area, it was critical to limit the scope of this initial work so that comprehensive and in depth solutions can be explored and compared to other existing well established methodologies. The work carried out in this study attempts to create the foundation upon which many critical topics related to the user attribution problem can be addressed via future studies. One future study with high applicability to the area of security is the topic of timeliness and concept drift as they relate to the recognition of a malicious user.

The goal of this work has been to provide a solution to the user attribution problem which can be measured in terms of its accuracy and the ability to maintain consistently high levels of identification precision when the number of users and/or web destinations visited increases. Chapter 2 provides more motivations for the existence and impact of the user attribution problem in the context of complex networks, while chapter 3 describes in detail the approach used to address the user attribution problem using hierarchical temporal memories.

Chapter 2

Review of the Literature

Solutions to the problem of correctly recognizing the unique source of communication have been proposed in the literature and range from IP Traceback techniques to web usage mining approaches. Unfortunately, as will be detailed in the following paragraphs, these approaches fall short of effectively addressing the problem.

One technique that has been proposed to recognize the source of communication is to trace the communication traffic back to its origin. Extensive IP traceback research is available that attempts to recognize the source of any packet sent over the internet. Traditional IP traceback attempts to address this problem in the context of security for DOS and DDOS attacks. A large number of IP traceback techniques have been proposed in the literature as reported by Santhanam, Kumar, and Agrawal (2006), all of them entail tracing backwards through routes taken by packets from the victim node, with trace traffic possibly passing through several ISP domains necessitating inter-domain cooperation. Typical challenges faced by IP traceback algorithms in recognizing the source of the attack range from having to address potentially forged (spoofed) source IP addresses, to attempting to trace attackers hiding behind stepping stones. These compromised hosts overwrite their source IP address on the outgoing packet headers and also apply header transformation to conceal their true origin. It is important to note that network address translation (NAT) devices, similarly to stepping stones, modify the private source IP address of packets by replacing it with the public IP address of the egress interface of the NAT device together with a potentially new port number. As pointed out by Santhanam et al. (2006), most IP traceback schemes are only capable of tracing up to the stepping stone but not

beyond and thus IP traceback schemes would be unable to detect sources behind a NAT device. Because IP traceback schemes rely on routing, they will not work across firewalls, unless the firewalls were to be specifically configured to allow traceback traffic. Individual organizations would find it difficult, if not impossible, to successfully utilize IP tracebacks without the involvement of the upstream ISP (Belenky & Ansari, 2003).

There are two major ways to trace back IP flows to their source; Reactive and proactive. Reactive approaches start tracing after the attack is detected while proactive methods log information for tracing while packets are in transit so that when tracing is required, the target of the attack can refer to this information to identify the attack source.

The reactive scheme of controlled flooding, as proposed in Burch and Cheswick (2000), is used to trace the source of DOS attacks and relies on the fact that during DOS attacks the links of the attack paths should be heavily loaded. By measuring incoming traffic to the attacked system and adding more traffic load to the links of the suspected path, attack packets to the target are expected to decrease as they are dropped. If this happens the process is repeated for the next hop until the source of the attack is detected.

Stone (2000) proposes a tracing overlay network "CenterTrack" where a tracing router is used to tunnel all monitored traffic through itself from edge routers. All traffic is monitored using signature-based intrusion detection and when an attack is detected the source is found to be only one hop away from the target. Chang et al. (1999) proposed Deciduous (Decentralized source identification for network based intrusion), which leverages knowledge of the network topology to establish IPSec tunnels between routers and the target. If an attack packet gets authenticated via a security association, then the attack originates at a point behind that router,

otherwise the attack source originates in the path between the router and the target. The schemes described have several limitations in that all require significant ISP cooperation for performing the trace back. CenterTrack suffers from scalability limitations based on the centralized forwarding approach and likewise IPSec is also not scalable because of the implicit processing overhead required to iteratively built security associations to investigate the integrity of links.

Probabilistic packet marking is used in (Savage, Wetherall, Karlin, & Anderson, 2001) where routers mark packets picked at random (by using the identification field of the IP packet that passes through them) with their own address so that when the target receives enough such packets it can reconstruct the addresses of all marking routers along the attack path. iTrace is used in (Bellovin, Leech, & Taylor, 2003), this software uses ICMP Traceback messages to help trace IP packets back to their source. When forwarding packets, routers can generate (with a low probability) a traceback message that is sent along to the destination. When enough traceback messages from enough routers along the path are available, the traffic source and path can be determined. The traceback message contains next and previous hop information together with as much of the traced packet as will fit. The attack path back to the source can be reconstructed using a time to live field available in the trace back message and the addresses of the routers on the attack path that implement iTrace. The Intension-drive iTrace method proposed by Mankin, Massey, Wu, Wu, and Zhang,(2001) represents an improvement over the original iTrace scheme in terms of reducing the number of iTrace messages that are not applicable to a specific sought attack and improving the time to complete trace backs.

Authors in (Snoeren et al., 2002) propose a scheme called source path isolation engine (SPIE) where each router, known as a data generation agent (DGA) saves partial information of

every packet that passes through that router so that it can be reused in the future to determine if that packet had passed through it. The challenge with this scheme is to minimize the amount of data that needs to be saved. The authors propose to save the IP header of packets and the first 8 bytes of the payload by hashing this data to produce several digests which are stored in a space-efficient data structure called a bloom filter, which considerably reduces storage requirements.

IP traceback schemes rely on routing to trace back the source of communication and thus require cooperation among network operators to support the specific tracing scheme requirements. When such cooperation is clearly defined, IP traceback schemes can be effective, but when such collaboration is not clearly outlined, then it would be difficult to trace users as they move across complex networks.

A different way to tackle the problem of identifying the source of communication is to monitor and track information that users access and the way in which they access it. That is, utilize past communication data that describes patterns of usage about users visiting a given web site, such as IP addresses, page references, date and time of access, in order to uniquely and repeatedly identify such users.

Web Usage mining is the process of applying data mining techniques to the discovery of usage patterns from web data, targeted towards various applications (Srivastava, Cooley, Deshpande, & Tan, 2000). These web mining techniques are applied mainly in the analysis of log based data and entail the following steps: pre-processing, pattern-discovery, and pattern-analysis. In the context of web usage mining, identifying user sessions is a challenge because of the difficulty of obtaining reliable usage data due to the presence of proxy servers, anonymizers, dynamic IP addresses, missing references due to caching, and the inability of servers to distinguish users during different visits to web sites.

Grčar (2004) describes web usage mining as the process of discovering usage profiles instead of user profiles. By studying web server logs, he came to the conclusion that sessions are easier to identify than users. However, in the worst case, the only user identification information included in a log file is the user source IP address. Grčar recognizes that the user's IP address is a poor form of identification since different users can be assigned the same IP address and one user can be assigned different IP addresses even during the same session. Other researchers make similar claims (Pitkow, 1997; Rosenstein, 2000). A technique proposed by Cooley, Mobasher and Srivastava (1999) to distinguish users with the same IP address is to make use of the user agent field and the HTTP referrer field. Cookies created by a given web site represent an even better form of user identification since 90% of users have cookies enabled (Baldi, Frasconi, & Smyth, 2003). Identification of the end of user sessions, as recommended in Cooley, Mobasher and Srivastava (1999), is carried out with the assumption that if a given predefined time period is exceeded visiting a web site or between two accesses to the same web site, the current session ends and a new session starts at that point. Because the sessions can have holes (missing web pages) due to the presence of web caches, the missing web pages can be inferred based on the site structure.

Sessions as proposed in (Chen, Park, & Yu, 1996; Cooley, et al., 1999), can be divided or joined into transactions. Transactions are made up of auxiliary web pages that are visited as part of the navigation toward a desired web page and content web pages that are the ultimate destination for users. Transaction identification which creates meaningful groups of references (URLs) for each user, is carried out using reference length (time spent by a user viewing a given web page) and maximal forward reference (the last page requested by a user before backtracking occurs). Grčar (2004) represents transactions as vectors of weights where each weight for a

given web page represents either the amount of time spent on that web page or the number of times that page is visited. The author measures similarity among transactions using the cosine similarity measure in order to cluster together transactions that belong to the same user.

Association rules represent a different approach to clustering also based on distance measures. The work of Cooley et al. (1999) shows that association rules utilizing the A-priori algorithm can be effective in the area of recommender systems. Frequent itemsets of visited web pages can be discovered which could show that web page-x and web page-y are accessed together 20% of the time. Association rules can then be used to show that when web page-x is accessed in a transaction, web page-y is also accessed a certain percentage of the time. This approach is promising but requires all data processed by the A-priory algorithm to be available, thus precluding application of the algorithm to data streams.

User session reconstruction is important in web mining activities and entails correct mapping of activities to different distinct users and the correct separation of activities belonging to different visits of the same individual. As users navigate a site, user identification occurs via identifiers such as cookies, source IP addresses and user agent fields, while session identification either utilizes embedded identifiers if available, or time heuristics or navigation heuristics which utilize the referrer field, so that a page must have been reached from a previous page to belong to the same session (Liu 2008). Identifying user sessions is similar to the problem of identifying individual users since references to web pages must be grouped into logical units representing web transactions or user sessions (Liu & Wu, 2004).

The authors in (Spiliopoulou, Mobasher, Berendt, & Nakagawa, 2003) tackle the problem of evaluating heuristics for session reconstruction and propose two key steps in this process:

1. All activities performed by the same physical person should be grouped together
2. All activities belonging to the same visit to the web site should be placed into the same group

The W3C (W3C Web Usage Characterization Activity 1999), defines *(server) session* or *visit* as the group of activities performed by a user from the moment he enters the site to the moment he leaves it. Since a user may visit a site more than once, the web server log records multiple sessions for each user. A user activity log, records the sequence of saved activities belonging to the same user. Thus, sessionization is the process of segmenting the user activity log of each user into sessions representing a single visit to that web site. Spiliopoulou et al., define proactive and reactive sessionization heuristics to perform such segmentation. Use of cookie-based identification and user authentication are considered proactive approaches since the mapping between the user and a session is guaranteed while (or even before) the user is accessing the site. On the other hand, reactive strategies like utilizing a source IP address and user agent, would attempt to establish such a mapping from the servers logs after the user has accessed the site. The authors conclude that use of cookies, coupled with time based heuristics (measuring session or inter user request timeouts) allows correct reconstruction of user sessions over 90% of the time. They further report that use of cookies improves the quality of reconstructed sessions by 20%.

Unfortunately, cookies have shortcomings because they cause privacy concerns, they are easy to remove from a user browser and they cannot be used across web site visits since each cookie is created by a specific origin server to be used only by a specific user. The authors (Iváncsy & Juhász, 2007) address the cross-site shortcoming of cookies by using two cookies, one (first party cookie) to track a user at one web site and the other (a third party cookie) using a

central server controlled by the authors. This third party cookie is created by the central server and remains the same across all sites visited during the test. The authors embed in all web pages of visited sites a reference to a small 1x1 GIF image residing in the central server, which when the web page is accessed, downloads the third party cookie. This cookie allows correlation of all sessions belonging to a given user across web sites. The authors show that their approach outperforms, in user identification accuracy, approaches that just use the source IP address. While the authors do manage to track user sessions across web sites, their experiment could easily fail if cookies were to be deleted by the user. More importantly, this experiment would also be difficult to implement in a real world scenario since this solution requires each site to embed the small image and deployment of a central server would likely suffer from scalability problems as the number of users increases.

Most often, the objective of user identification is to recognize the user across repeated entries to multiple web sites, without implicit identifiers (cookies) or explicit identifiers created when users login or register to access sites. Without such identifiers, accurate user identification is a challenge and user identification must then be inferred which turns the user identification problem into a user attribution one. A new approach is needed that could recognize users accessing a multitude of web sites without having to rely on cookies.

In (Jin, Sharafuddin, & Zhang, 2007) the entropy of the persistence of IP addresses is computed but the objective of this study is to identify the presence of dynamic IP addresses rather than profiling single users. User profiling at the granularity of single users was studied in (Song, Venable, & Perrig, 1997) for user recognition, by monitoring keystroke latency patterns or at the device level, for device recognition by fingerprinting devices via detection of changes in clock skews among different devices using the TCP Timestamp option (Kohno, Broido, &

Claffy, 2005). 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. The time stamp option defined in (Jacobson, Braden, & Borman, 1992) shows promise when present in TCP packets for improving the user session identification process. A way to utilize this optional TCP field is utilized in the approach section of this idea paper.

Tracking electronic identities in communication networks can be achieved by using "signatures" of node activities (Cormode, Korn, Muthukrishnan, & Wu, 2008). Signatures capture the distinctive and discriminating communication behavior of an individual. The authors adopt a signature based approach to analyze the patterns of communication exhibited by individuals. Using real data the authors measure key signature properties in the form of persistence, uniqueness and robustness in order to detect, among several scenarios, label masquerading. Label masquerading occurs when a user switches all of his communication from one node to another. An example of this, is the repetitive debtor problem (Hill, Agarwal, Bell, & Volinsky, 2006), where a consumer switches accounts with no intention of paying for his network usage. In their experiments, the authors found that high persistence (signatures remain stable across time) and uniqueness (signatures from one user should not match signatures from a different user) are key properties needed to correctly identify users that leverage label masquerading.

The idea of profiling and recognizing users based on their communication behavior was recently undertaken by (Kumpošt, 2007; Kumpošt & Matyáš, 2009). The authors use real data from a university campus network to recognize users based on their IP address and

communication profile using SSH, HTTP and HTTPS. The experiments conducted by these authors produced reasonably accurate identification results for SSH type traffic with a 21% rate of false alarms; however for HTTP and HTTPS traffic the false alarm rate was high, 70% and 60% respectively. The authors attributed the poor performance to the fact that students connecting to the internet utilized wireless connections from laptops from multiple locations across campus and therefore were assigned different IP addresses.

Similar work on user profiling was also carried out by Yang (2010). Her approach was to build user profiles of web browsing behavior from consecutive web sessions of known users and use these profiles to predict the owner of future anonymous web sessions (i.e. user identification). The experiments conducted by the author show that this approach can be highly effective and efficient. However, the solution does not scale well (the experiment could not go beyond 100 users) since many users can share the same behavior and her approach cannot connect consecutive user sessions, forcing identification to take place over short periods of web activities. Yang acknowledges that recognizing web users based solely on their online user behavior rather than using tracking techniques is a difficult problem. The problem is made even more complicated due to the basic need of recognizing a user when that user identity is not known in the first place. This is very important for real life applications where supervised learning (assuming the user is known at training time) cannot be applied.

Herrmann, Gerber, Banse & Federrath (2010) implement web user identification attacks by linking web sessions of a given user solely based on the history of his past activities on the web and specifically by observing how frequently different host names are visited by users. Their experiments are limited to 28 users and show that consecutive sessions can be linked to a

given user with a high probability for session durations ranging from 5 minutes to 48 hours. Their results show correct user identification for 50% of the users 80% of the time.

The recent work of both Yang (2010) and Herrmann et al. (2010) shows promise in utilizing approaches that leverage user web past activities to identify users. Their experiments share limitations that have left open opportunities to perform more research in this area:

- Both leverage supervised learning and assume that the user is known at training time
- Both do not address concept drift and are unable to adjust to changes in users' behavioral patterns
- Both are limited in their ability to identify user sessions. Yang's approach attempts to match sessions learned from user web activities before time *T* with anonymous sessions observed after time *T*. Moving away from time *T* increases the difficulty of connecting sessions belonging to the same user. The approach used by Herrmann et al. (2010) uses a fixed time window (all activity falling within the window belongs to the same user) to group sessions belonging to the same user. This solution will erroneously distribute contiguous sessions belonging to the same user to a new user.
- Both solutions do not scale in real world situations as user identification relies exclusively on the uniqueness of the web destinations visited by users, as is acknowledged by Herrmann et al. (2010) : "…. *tracking one user among thousands of unknown users will cause a false alarm for the majority of instances*…"

Even more recently, Banse, Herrmann, and Federrath (2012) address the challenge of tracking internet users (linking a large number of multiple user sessions) without resorting to the use of explicit tracking techniques such as cookies or other explicit identifiers. The authors also

explicitly address the problem of changing source IP addresses in their solution by assuming that addresses can change only within a fixed time period (24 hours), an epoch. In their experiments, the input is represented as a triplet: epoch, source IP, destination IP. Session identification is accomplished by aggregating all events that share epoch and source IP. The authors verified their solution in a real world setting with up to 2000 concurrent active users each day and report being able to correctly link up to 88% of all sessions on a day by day basis. It is important to note that the research left as future work by these authors is tackled by the work carried out in this study, namely:

- The authors acknowledge that NAT devices which force the same source IP address to be shared among multiple users, is a problem not addressed in their solution
- The authors acknowledge that increasing the number of times that the source IP address changes from once a day to every three hours decreases the accuracy of their algorithm from 60% to 49%
- Epochs as defined in this work are tied to a fixed location where user requests are issued. In order to address mobility the epoch would need to account for location (time zone)

As this literature review has shown, the problem of recognition of users that get assigned different source IP addresses re-entering the network or accessing the network from multiple locations is a real issue that needs to be addressed especially in the area of security for intrusion response and prevention systems. The research in this topic, which has leveraged trace back, tracking and inference techniques, has so far not provided an effective solution to this problem

Complex Networks

A system can be complex and complicated at the same time but these are two very different concepts. Barrat et al. (2008) explain that the intricate appearance of large-scale graphs naturally evokes the idea of complicated systems in which large number of components work together to perform a function. The internet is a physical system that is composed of independently administered computer networks each of them having its own administration, rules and policies. There is no central authority that oversees its growth as new connections and nodes are added to it on a daily basis. These attributes make the internet a complicated system, much like mobile networks that attach to the internet from a multitude of different locations. Yet, the internet and mobile networks also share properties of complex systems where they dynamically evolve and self organize in very specific structures that maintain a scale-free topology.

Typically, complex networks are difficult to describe based on their topology. Many of them form networks whose vertices are the elements of the system and whose edges represent interactions among them. In the case of the World Wide Web, vertices are HTML documents connected by links pointing from one page to another, while in the case of the internet vertices are routers and computers linked by various physical or wireless links. Because of their large size and the intricacy of the interactions, the topology of these networks is largely unknown (Barabasi & Albert 1999). Barrat et al. (2008) believe that complex systems consist of a large number of elements capable of interacting with each other and their environment in order to organize in specific emergent structures. These authors attribute the characteristics of complex networks to the fact that decomposing the system and studying subparts in isolation does not allow an understanding of the whole system and its dynamics, since the self organization principles reside

mainly in the collective and unsupervised dynamics of the many elements. Mobile networks, in the form of WIFI hotspots and cellular networks provide access to the internet and support connections to the World Wide Web. Mobile networks add to the intricacy of the topology of the internet in many cases with deployments where one wireless network overlaps with another, with different operators administering these networks and different standards that define how these networks attach to the internet.

The patterns of connections among elements of complex networks are neither regular nor random, instead these networks tend to self organize into a scale free state where the probability P(K) that a vertex in the network interacts with other K vertices decays as a power-law, following $P(K) \sim k^{-y}$ where *y* is a constant. The power law tail characterizing P(K) indicates that highly connected (large in-degree) vertices have a large chance of occurring, thus dominating connectivity. Barabasi and Albert defined the principle of "preferential attachment" as typical of complex networks, where there is a higher probability that a new vertex will be linked to another existing vertex that already has a large number of connections. This power law behavior strongly contrasts with the Poisson degree distribution of classical random graphs where links are randomly created between pairs of existing nodes.

One key property that is traditionally shared by complex networks is the fact that while complex networks are often large in size, in most networks there is a relatively short path between any two nodes. This is known as the concept of "small worlds". Broder et al. (2000) found that the average path length between nodes in a 50 million node sample of the World Wide Web is 16, while Adamic (1999) found that for 60,000 web root nodes the average path length was 3.1 hops, leading him to acknowledge that the World Wide Web is a small world. Another important property of complex networks is the high tendency for nodes to cluster together so that

nodes will tend to create tightly knit groups (cliques) characterized by a relatively high density of ties. The local clustering coefficient of a vertex (node) in a complex network quantifies how close its neighbors are to being a clique (complete graph). Watts and Strogatz (1998) defined this coefficient to determine whether a graph is a small-world network.

As previously described, the degree of a vertex in a network is the number of edges connected to that vertex. P(k) was previously defined to represent the fraction of vertices in the network that have degree k and it follows a degree distribution that has a power tail. Both the World Wide Web and the internet follow two power law degree distributions; $P_{out}(k) \sim k^{-Yout}$ that describes the probability that a node/document has k outgoing edges/hyperlinks and $P_{in}(k) \sim k^{-Yin}$ that describes the probability that k edges/hyperlinks point to a certain node/document. Different studies for the World Wide Web show values for Y^{out} ranging from 2.45 for document sizes of 325729 to 2.72 for document sizes of $2 \text{ X } 10^8$.

Of specific interest to this study is the evaluation of the distribution of visitors to web sites. Adamic and Huberman (2000a) 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 (2000a) find inconsistencies in the conclusions of a study by Barabasi and Albert (1999) 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 (2005) 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 at its fullest 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, few and unique 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 assumption 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.

In order to leverage the power law properties that characterize users' web site visits, this research created synthetic data for its experiments by implementing a zipf generator that simulates user visits to web sites ranging from 1 to N, with web site 1 being the most popular and N the least. The zipf distribution is of the form:

$$\text{Zipf(n)} = \frac{C}{n^{\theta}} \quad \text{where C} = \left[\sum_{i=1}^{N} \left(\frac{1}{i}\right)^{\theta}\right]^{-1}, \text{ N = maximum number of web sites,}$$

and $0 < \theta < 1$. The algorithm used to implement this distribution are based on zipf algorithm used in (Gray, Sandaresan, Englert, Baclawski, & Weinberger, 1994) as shown in the Java snippet in Figure 1. *Next_ZipfRandom* 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 below, 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). To see how the *Next_ZipfRandom* function is used in the context of the research experiment, refer to section "Generation of Synthetic Data for the Simulation".

It is important to note that, 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 (2003) 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 the HTM to correlate groups that are close in space and time.

```
long  Next_ZipfRandom(long n, double theta)
{
  double   alpha     =   1.0 / (1.0 - theta);
  double   zetan     =   zeta(n,theta);
  double   eta       =  (1.0 - Math.pow(2.0 / (double)n, 1.0 - theta)) / (1.0 - zeta(theta,2.0)/ zetan);
  double   u         =  random.nextDouble(seed);
  double   uz        =  u * zetan;

          if(uz  < 1.0)  return (1);
          if( uz < 1.0 + Math.pow(0.5,theta)) return (2);
          return(1 + (long)(n * Math.pow(eta * u - eta + 1.0,alpha)));
}

long zeta(double n, double  theta)
{
 int i = 0; long ans = 0;

          for(i=1;  i< n; i++)
          {
         ans +=  Math.pow(i+1,theta);
      }
      return(ans);
}
```

Figure 1 Simulating user web visits to web sites using Zipf distribution

Chapter 3

Methodology

As humans generate more and more data in their lives, they leave behind massive amounts of information that reveal their unique behavioral characteristics. Using this data, it is possible to recognize each user. User classification is the process of matching records of past behaviors that belong to the same individual, sometimes when the individual is acting anonymously**.** Hills and Nagle (2009) define identification in the context of dynamic networks as a matching task that involves comparing network entities across time periods. The authors acknowledge as a limitation the theoretical aspect of their work in modeling real user behavior. Kumpošt and Matyáš (2009) tackle this limitation by addressing the user attribution problem using an identification approach which pinpoints users among others based only on observed behavioral characteristics. The approach that has been used in this study to address the user attribution problem relies on the premise that users follow patterns of behavior peculiar to them and is reflected in a time ordered set of unique destinations visited during communication sessions. Specifically, each communication source visits frequently and persistently over time unique destinations with respect to other communication sources. These destinations are visited in a specific order in the context of user sessions. Observing user sessions over time, together with the order of visits to specific web sites, can be leveraged to infer unique users re-entering the network.

There are two important requirements that this solution to the user attribution problem has addressed:

1. Accurate recognition of users re-entering the network with potentially new source IP addresses
2. The ability to recognize communication patterns belonging to the same user (among many different user sessions belonging to many users) by analyzing the sequential time ordered nature of web visits, much as is done in web usage mining to predict what web page a user clicks next (Liu 2008). However, as opposed to traditional web usage mining, user sessions are be tracked across many different web sites.

Network signatures, derived from user navigational patterns, have been shown to be effective for targeted marketing and advertising in identifying online users based on their browsing behavior (Hill, et al., 2006). In addition, social network signatures have been used for author attribution of written documents, where the identities of authors of articles can be inferred based on the authors they cite (Hill & Provost, 2003). The applications of re-identification are vast, ranging from protecting the privacy of personal records to asymmetric threat detection for national security, to detecting subscription fraud in the telecommunication industry. Development of reliable methods which forgo use of tracking techniques such as cookies, logins and keys and only rely on web usage patterns for identification of users in communication networks is an important problem (Yang 2010).

Hill et al (2006) show that the level of activity in connections among nodes and the freshness of such connections can be used to predict node behavior in dynamic networks. The authors used communication signatures derived from the levels of activity of nodes and edges in graphs to detect repetitive fraud by users re-entering the network with new IDs and to recognize users' repeated access to web servers. The authors represented the evolution of network transactions over time among nodes by tracking: (1) Lifetime of node relationships, (2)

Frequency of transactions among nodes, (3) Degradation in the relative importance of the relationship with the passing of time.

The experiments conducted by these authors provide the motivation for this study by showing that by monitoring the evolution of network transactions, predictive performance of user behavior continues to significantly improve for user connectivity to web sites as the number of connections increases. They conclude that despite the increase in the number of connections, the predictive performance improves, thus allowing a more representative signature to be built for each user. It is important to note that while addressing the user identification problem, the authors do not explicitly deal with the problem of identifying subscribers re-entering the network when the source IP address changes due to dynamic re-assignment or due to the presence of NAT or web proxies.

A solution to the user attribution problem as implemented in this study addresses both the spatial and the temporal aspects of the network data used as input. The spatial aspects are tied to the recognition of unique user sessions and unique destinations visited by a user. The temporal aspects are tied to the need to observe time ordered visits to different web sites in the context of a user session in order to better discriminate among multiple users.

Hierarchical Temporal Memory (HTM) is a technology which is modeled on the algorithms used in the neocortex of the brain (George & Widrow 2008; Hawkins, George, & Niemasik, 2009). Network nodes in an HTM, are organized in a hierarchical way, with each node implementing learning and memory functions. HTMs are unique in stressing the temporal aspect of perception, 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 (Duch, Oentaryo, & Pasquier,

2008). 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) (Greff, 2010).

The predictive power of HTMs comes in part from their use of Markov models in the context of Bayesian networks used to propagate beliefs across the hierarchy. Markov models as proposed by Deshpande and Karypis (2004) are well suited to address the temporal aspect of the inference problem and have traditionally been proposed as the underlying modeling machinery for web link prediction and web pre-fetching to minimize system latencies. The ability of HTMs to infer causes of novel inputs in space and time and to make predictions leveraging the hierarchical nature of the input data, makes HTMs good candidates to be used to address the user attribution problem.

HTMs have been successfully used in classification problems in a variety of applications. Experiments conducted by Bobier (2007) showed recognition accuracy of 95% by using the commercial Numenta's NuPIC framework to model HTMs in the context of recognition of USPS handwritten digits. Besides being applied to image recognition, HTMs have also been applied to speech recognition with promising results as reported in the work of Doremalen and Boves (2008). HTMs have also been used to model and predict user choices. In (Melis, Chizuwa, & Kameyama, 2009), the authors build a mobile phone intention prototype using HTMs and Bayesian Networks to predict user intentions while using a mobile phone based on the menu choices that the user selects. The authors report that the HTM performs well and is able to easily

use information (input) from the real world with little preprocessing and good accuracy. The authors conclude that when the structure of the application is reflected into the HTM, even better results can be obtained, a theme that is consistent in the literature.

HTMs have also been used in the area of web analytics. In a talk given for the association of computing machinery (ACM), 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) 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 allows prediction to fully leverage the dynamic "context" (patterns embedded in the sequence of the most often visited web pages) of web pages visited by a user.

The user attribution problem implemented in this work benefits from the use of variable Markov models. These models increase (over fixed order Markov models) the predictive power of HTMs and are critical in enhancing the accuracy of proactive identification of recurring temporal patterns (visits to web sites). This study has utilized a three node HTM, as shown in Figure 2, that was used to classify unique users re-entering the network with the bottom node recognizing user navigational patterns and the top nodes recognizing user sessions (higher level concepts) for each properly classified user.

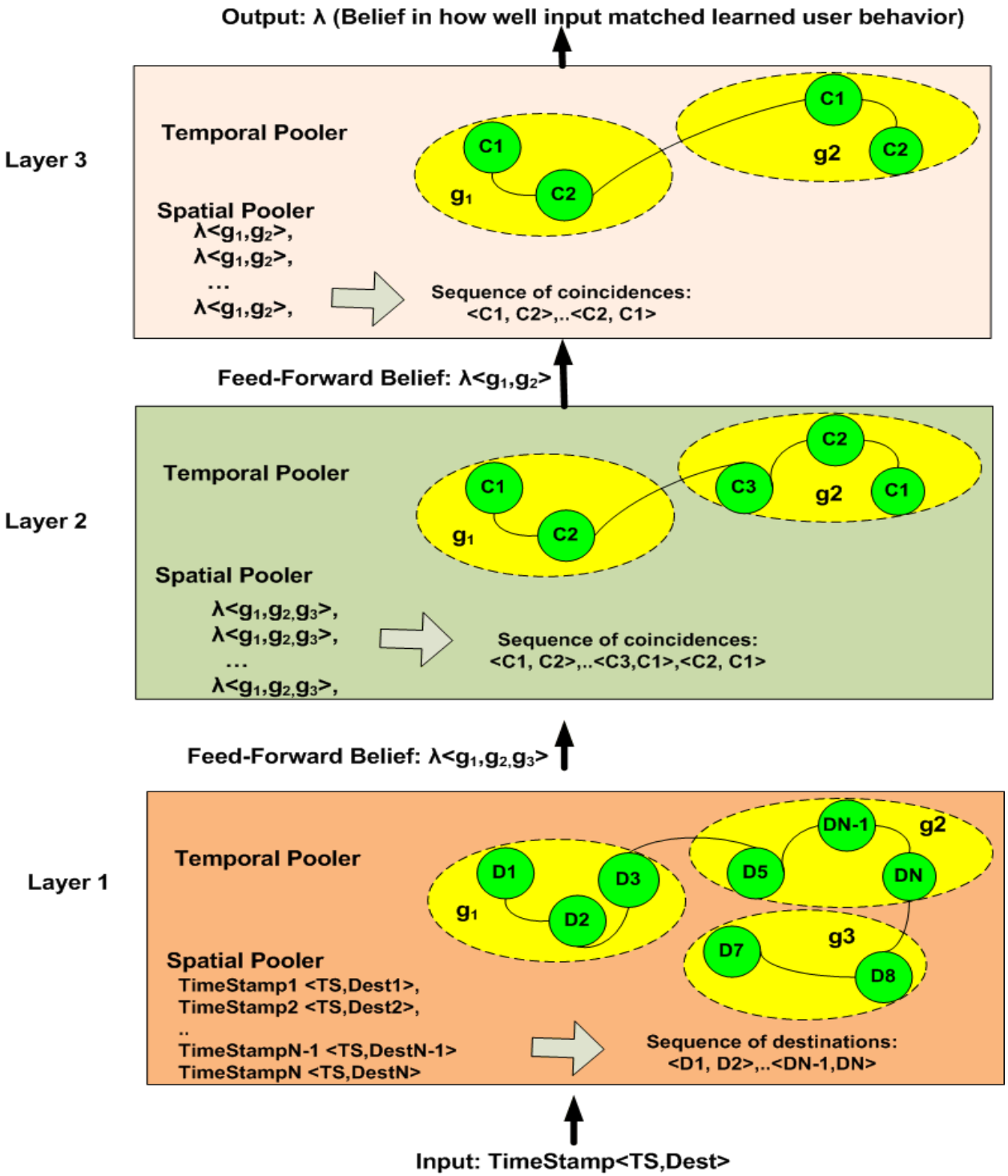


Figure 2 Three Layer HTM for User Attribution

The HTM at layer 1 collects sequence of web destinations and maintains their temporal relationships (navigational patterns that are likely to follow each other for a given user) via a Markov graph. These destinations are then be broken up into separate temporal groups based on their sequential relationships and the connectivity strength of the connections among the destinations. One can think of these destinations as representing different areas of interest for a given user (e.g. soccer and tennis for group 1, high school and universities for group 2 and blogs and movies for group 3). However, layer 1 would not deal with the temporal relationships among the temporal groups it creates. Creating temporal relationships among groups created in layer 1 is the job of layer 2 of the HTM.

The HTM at layer 2 deals with relationships among higher level concepts received from layer 1 in the form of temporal groups. For instance, temporal group 1 from layer 1 (a coincidence in layer 2) can be thought of as representing the higher level category of sports, while group 2 from layer 1 could represent education and group 3 could represent entertainment. Layer 2 learns navigation patterns among these higher level concepts received from layer 1.

Layer 3 is similar to layer 2 and learns temporal relationships among the higher level concepts received from layer 2. So, for instance, for a given user, layer 3 could have learned that the user navigates sports followed by education and then entertainment sites. Each layer of the HTM communicates to the layer above the degree to which the input is similar to temporal patterns (feed forward beliefs) learned within that layer. Layer 3 is then responsible for generating the final output in the form of a belief in how well the input matches learned navigation patterns for this user inferred through the three layers of the HTM.

The generalization property brought about by the hierarchical structure of HTMs has another advantage in that it enhances the ability of HTMs to correctly recognize ordered visits to web sites by different users. When users visit web sites and the corresponding ordered site visits are stored in a Markov chain, it is critical to be able to distinguish the start and end of a sequence of such visits. A predefined amount of time between visits is used to mark the beginning and end of such visits. However, if web site visits by a given user fall outside this time window then the original sequence is viewed as a set of many smaller sub-sequences, possibly with only one element in each. Under these conditions a Markov chain would lose accuracy by not being able to operate on a longer sequence. In an HTM, higher layers of the hierarchy are able to recover, to a large extent, the original sequence and thus improve the accuracy of the HTM. The hierarchical structure of HTMs increases the discrimination power of the model by improving the ability to recognize long recurring patterns while at the same time becoming less susceptible (more invariant) to the time differences in the arrival of input. Riesenhuber and Poggio (1999) originally proposed a model for visual processing in the cortext as a hierarchy of increasingly sophisticated representations. This hierarchical model was consistent with physiological data from inferotemporal cortex that accounts for the complex visual task and makes testable predictions. They obtained invariance in the model (to changes in the position of an optimal stimulus) by generalizing simple cell to complex cell relationship by using a maximum operation (max) performed on the simple cell inputs to the complex cells, where the strongest input determines the cell's output. This preserved feature specificity. The model also alternated layers of units combining simple filters into more complex ones to increase pattern selectivity with layers based on the max operation. This hierarchy helped to build invariance to position and

scale while preserving pattern selectivity. A similar approach has been used in this study to combine outputs from different layers of the HTM.

Tremendous and potentially infinite volumes of data streams are often generated by real-time internet traffic. Unlike traditional data sets, stream data flow in and out of a computer system continuously and with varying update rates. They are temporally ordered, fast changing, massive, and potentially infinite. For example, the universe corresponding to the set of all pairs of IP addresses on the Internet is very large, which makes exact storage intractable (Han & Kamber, 2006). This study assumes that data is not collected from data bases or servers (web proxies/servers) as traditionally done in web mining; instead it assumes that data is processed "off the wire". This requirement is due to the need to collect the TCP time stamps (the need for this parameter is explained later in this paper) from the data stream which is not normally found in web server log files. Hierarchical temporal memories were chosen for this study because of their ability to process (learn and infer) streams of data (George & Widrow, 2008). As described in the next sections, not all approaches are well suited to deal with stream data.

Methodologies for stream data processing address the need for infinite amount of storage space to store streams and often settle for approximate rather than exact answers. Synopses provide summaries of stream data, which typically can be used to return approximate answers to queries. Random sampling, sliding windows, histograms, multi resolution methods (e.g., for data reduction), sketches (which operate in a single pass), and randomized algorithms are all forms of synopses (Han & Kamber, 2006).

Traditional methods of frequent itemset mining, classification, and clustering tend to scan the data multiple times, making them infeasible for stream data. In addition, these techniques

ignore the temporal order in which transactions occur (e.g. the order in which web pages are visited). Stream-based versions of stream data mining instead try to find approximate answers within a user-specified error bound. Examples include the Lossy Counting algorithm for frequent itemset stream mining as described in Manku and Motwani (2002), which divides the incoming stream of data into buckets, computes the approximate frequency of items accounting for maximum frequency error and keeps only items in buckets that are most frequent. This simple approach unfortunately suffers from at least two short comings: (1) the frequency list of itemsets in each bucket may grow infinitely as the stream goes on; (2) frequent itemsets are scanned many times impacting the efficiency of the algorithm. Another example of an algorithm used for stream data classification is the Hoeffding tree (Domingos & Hulten, 2000; Hang & Fong, 2010). This algorithm which was originally used to track web clickstreams, uses decision tree learning and creates nodes incrementally as more data streams in. An advantage of Hoeffding trees is that this algorithm does not scan the same data multiple times and can classify data even while the tree is being built. A disadvantage of this technique is that it cannot handle concept drift (changes in the variables that are being classified) because once a node is created it cannot be changed. The implications of implementing concept drift in the HTM via continuous learning (learning occurring after training completes) have been explored with initial positive results, however, in order to limit the scope of this study continuous learning is left as an area of future study.

The Very Fast Decision Trees (VFDT) makes modifications to the Hoeffding tree algorithm to improve both speed and memory utilization but still cannot handle the concept drift in data streams (Domingos & Hulten, 2000). The Concept Adapting Very Fast Decision Trees (CVFDT) addresses the concept drift by staying current in spite of continuously changing data

by growing an alternate sub-tree whenever an old one becomes questionable and replacing the old one with the new one when the new one becomes more accurate (Hulten, Spencer, & Domingos, 2001). One of the shortcomings of the previously described techniques is that they ignore the temporal ordered (sequential) nature of the data stream, an important aspect of the approach chosen for this study that is needed to further improve the accuracy of user recognition.

Sequential pattern mining is the mining of frequently occurring ordered events or subsequences as patterns. Given a sequence database, any sequence that satisfies minimum support is frequent and is called a sequential pattern (Han & Kamber, 2006). An example of a sequential pattern in the context of web mining is "70% of users who first visit web page A.html and then visit web page B.html, in the same session, have also accessed web page C.html". Algorithms for sequential pattern mining include GSP, SPADE, and PrefixSpan, as well as CloSpan (which mines closed sequential patterns). The problem of mining sequential patterns was first proposed by Agrawal and Srikant (1995). In the Apriori-based GSP algorithm, Srikant and Agrawal (1996) generalized their earlier notion to include time constraints, a sliding time window, and user-defined taxonomies. Zaki (2001) developed a vertical-format-based sequential pattern mining method called SPADE, which is an extension of vertical-format-based frequent itemset mining methods. PrefixSpan, a pattern growth approach to sequential pattern mining, and its predecessor, FreeSpan, were developed by Pei et al. (2001) and Han et al. (2000). The CloSpan algorithm for mining closed sequential patterns was proposed by Yan, Han, and Afshar (2003).

Constraint-based mining of sequential patterns is another approach to mining sequential patterns which incorporates user-specified constraints to reduce the search space and derive only patterns that are of interest to the user. Constraints may relate to the duration of a sequence, to an

event folding window (where events occurring within such a window of time can be viewed as occurring together), and to gaps between events. Pattern templates may also be specified as a form of constraint using regular expressions (Han & Kamber, 2006).

Markov models have been proposed as an underlying model for web link prediction as well as web pre-fetching to minimize system latencies (Mukund & George, 2004; Sarukkai, 2000). These models represent web pages as states and transition probabilities represent the likelihood that a user will navigate from one state to another. Markov models which include Markov chains are especially suited for predictive modeling based on contiguous sequences of events. HTMs incorporate Markov models to recognize temporal patterns, as a key element of their architecture.

An important part of the approach utilized in this study hinges on the ability of the algorithm to recognize user sessions. That is, identify a set of HTTP requests bound for different destinations as a group of messages originating from the same source. The approach used for this study during the trainign pahse has utilized the TS value of the TCP timestamp option field of a TCP packet carrying an HTTP request as defined in (Jacobson, et al., 1992). The 32 bit TS value contains a counter that is driven by the clock of the originating device. This counter on most systems resets to a fixed value or to some random value when the device is rebooted. A consistent property of this field, needed to compute round time trip delays, is that it continues to increase over fixed time periods from some initial value until the device reboots or the TS counter wraps around. In this study, any new HTTP request belonging to a currently tracked user session with a TS value that falls outside a predefined positive sliding window is to be deemed to belong to a new user session, otherwise the request belongs to the user session currently being tracked. This algorithm, by utilizing the TS value, gains in discriminating

accuracy since it is able to *connect* sessions belonging to the same user. Contrast this with the work by Yang (2010) which relies on the identifying user sessions exclusively by relying on the destinations visited by the known user. Yang's approach limits the scalability of the solution, as acknowledged by the author, since "*as the number of users increases user identification based on behavioral patterns alone becomes infeasible…there is a need for methods (e.g. IP addresses, Cookies) to connect consecutive sessions. In cases where this assumption does not hold, identification can only be done on fairly short periods of web activity, which can be quite difficult….".*

In this study the TCP time stamp (TS) is used to identify and track only user sessions only during the training phase of experiments. 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. 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). 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 as shown in Figure 8 and only one HTM will be able to recognize it better that the other HTMs based on its past training.

The outcome of session identification determines the quality of the input received during the training and inference stages. Successful session identification would allow identification of

multiple web visits from a single source as belonging to a single unknown user, leaving the task of recognizing the actual user to the HTM Markov chain inference engines. On the other hand, unsuccessful session identification would either classify *multiple* web visits from multiple sources as belonging to a single unknown user or during training misclassify web visits from a single source as belonging to the wrong user. These conditions of course, would compromise the training and inference processes, making it impossible to correctly identify this web traffic as belonging to the correct user. This is exactly the type of session miss-classification problem acknowledged by Yang (2010) in her experiments.

The TCP time stamp was proposed (Jacobson, et al., 1992) to enable real time round trip time measurements between TCP peers and to protect against wrapped TCP sequence numbers in very high speed networks that use very large window sizes (greater that 64K bytes in size). In this study, the TCP timestamp update rate from a sequence of TCP packets have been used to fingerprint a user session. The TCP TS value is not a timer, but it represents an infinite counter started on a given device that is incremented typically every millisecond driven by the device internal clock and never stops incrementing as long as the device is powered on. Powering off the device will restart this infinite counter. This counter is sent in each TCP packet to a specific TCP peer that can use it as a synchronization point.

In this study during training, once an HTM has received the first TCP timestamp, it uses it to start its own infinite TS counter using its own internal clock and thus synchronizes with the originator of this TCP session. Because both the originator and the HTM use clocks indirectly to synchronize, the session identification algorithm utilized in this study needs to account for clock skew between the HTM clock and the clock of originator of this TCP session. In this research, a fixed window is used to measure possible clock skew. Unfortunately, using a fixed 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 approach could address the potential increase in clock skew that occurs over time between user and HTM overcoming the limitations of a non-resettable HTM TS counter. With this approach, it is possible to use a small window size since the algorithm is able to adjust to clock skew over time. The merits of this approach as well as determining the best size for this 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 over real mobile networks.

Clock skew is not the only way in which communication traffic belonging to a given user session can be misclassified. Based on radio frequency (RF) conditions, communication over mobile networks can suffer from elevated levels of noise with resulting high levels of data loss. The user session classification algorithm proposed in this study would need to account for possible invalid or outdated timestamps. For instance, TCP timestamps that fall within windows belonging to more than one HTM are obviously invalid and can be discarded. However, retransmitted TCP packets belonging to an old (no longer active) user session could potentially be misclassified as belonging to another active user.

This study plans to track multiple users visiting multiple destinations. The advantage of using the TCP time stamp field for session identificatin is its ability to track a given device that sends multiple HTTP messages to multiple destinations. This benefit is not provided when utilizing “cookies”, another user identification method also recommended in the literature for HTTP traffic. Cookies are not designed to track users across multiple destinations because they are created by each origin server to identify a given user. Instead, cookies work well when used as user identifiers to track multiple users all visiting the same web site. The referrer field of HTTP messages has also been used to track user sessions in web mining applications. This HTTP header reports the web page that was visited just prior to the current one and is used to create an ordered list of web pages visited at a given web site. The referrer field was not used in this study because client or web proxy caching can often result in missing access references due to pages or objects that have been cached (Liu 2008).

In the area of data preparation for web mining, Kumpost (2007) has used both TF-IDF and cosine similarity, techniques borrowed from text mining, to build user profiles from network traffic log processing. Kumpošt and Matyáš (2009), extended their work and, similarly to this study, looked into the issue of profiling and user recognition based exclusively on user past activity. The experiments conducted by these authors produced reasonable accurate identification results for SSH type traffic with a 21% rate of false alarms; however for HTTP and HTTPS traffic the false alarm rate was high, 70% and 60% respectively. The authors attribute the poor performance to the fact that students connecting to the internet utilize wireless connections from laptops from multiple locations across campus and therefore get assigned different IP addresses. The authors dealt with the spatial aspects of the identification problem and relied on a fixed two-dimensional matrix to represent communication between sources (rows) and destinations

(columns) with each row identified by the source IP address. In their approach, the source IP address is used to assist in the user attribution process.

The approach utilized in this study addresses directly the attribution problem in the context of changing source IP addresses by forgoing use of the source IP address in the attribution algorithm. This study also focuses on the user session identification problem, a problem that does not exist if one assumes that the source IP address does not change and thus can be used to uniquely recognize a given user. This research extends the work of Kumpošt and Matyáš in the area of user attribution in network communication in situations where the source IP address assigned to users or devices can change.

The solution leveraged in this study addresses directly the problem of user recognition by going beyond the use of spatial algorithms as proposed in Kumpošt and Matyáš's work, and instead it proposes both spatial and temporal algorithms in the form of HTMs which leverage the hierarchical properties of network data in order to anticipate learned user navigational patterns to allow more accurate identification of users re-entering the network.

Approach Introduction

The approach followed in this paper is based on the work of George and Widrow (2008), with the following key extensions:

1. This implementation deals with sequences of input. Markov graphs are used for both training the hierarchical temporal memory (HTM) and for performing inference. In the work of George and Widrow (2008), Markov graphs were only used during training and during inference a 0^{th} order Markov graph was used instead (each

element is considered to be independent of the previous one), which means that inference did not rely on the sequential order of input.

2. This implementation uses variable Markov chains and state cloning as a way to improve the learning and inference accuracy of the HTM.
3. This implementation leverages the idea of "playback" as a way to bootstrap and optimize learning across multiple layers of the HTM. This concept is missing in the work of George and Widrow (2008).

The user attribution problem as addressed in this study is really made up of two sub problems:

1. Communication session identification
2. Communication pattern identification associated with the same user communication session

Communication session identification entails recognizing multiple consecutive web destinations as being visited by the same user over time. Session identification is critical during the learning stage of the classification process to accurately train the HTM to correctly identify web sites visited by a user during the inference stage. The key element that enables identification of a source of communication as utilized in this study is the Time Stamp (TSval) value representing the timestamp option field of a TCP packet as defined in RFC 1323.

Beacken et al. (2011) 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 time stamp value, a 32 bit value

implementing a virtual clock on each device, is used to uniquely identify unique sessions associated with a given user. Each device implements the virtual clock as a 32 bit wrap around counter which starts at a fixed or random time (depending on the device) and is incremented at each tick time (RFC 1323 recommends the clock frequency to be in 1 *ms* as this forces a wrap around each 24.8 days). For this prototype a resolution of 1 ms was assumed, realizing that the actual clock resolution could also be extracted from the communication data itself. The chance of two devices having the same TS value is rare; theoretically it is $1/ 2^{32}$ assuming randomness since devices start their virtual clock at different times based on when the device is powered on.

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. All communication input associated with a given <TS> value within a given time window was 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) visited by this user. The input to the prototype has the following form:

Timestamp <*TS, Dest*>

The timestamp has a resolution of 1 millisecond and represents the passage of time with respect to the arrival of input to the prototype. 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 below was used for communication session identification during the learning phase of classification and selection of appropriate HTM to perform communication pattern identification.

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 with existing device type already identified

- 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 3 Communication Session Identification Algorithm during Training

Note that 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 windows 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}.

It is important to note that in this implementation, the same user utilizing two different devices would be identified by the prototype as two different users.

The HTM Implementation

The HTM implementation is shown in Figure 4 and depicts for each layer the different stages that the HTM goes through to learn new input patterns and then perform inference on them.

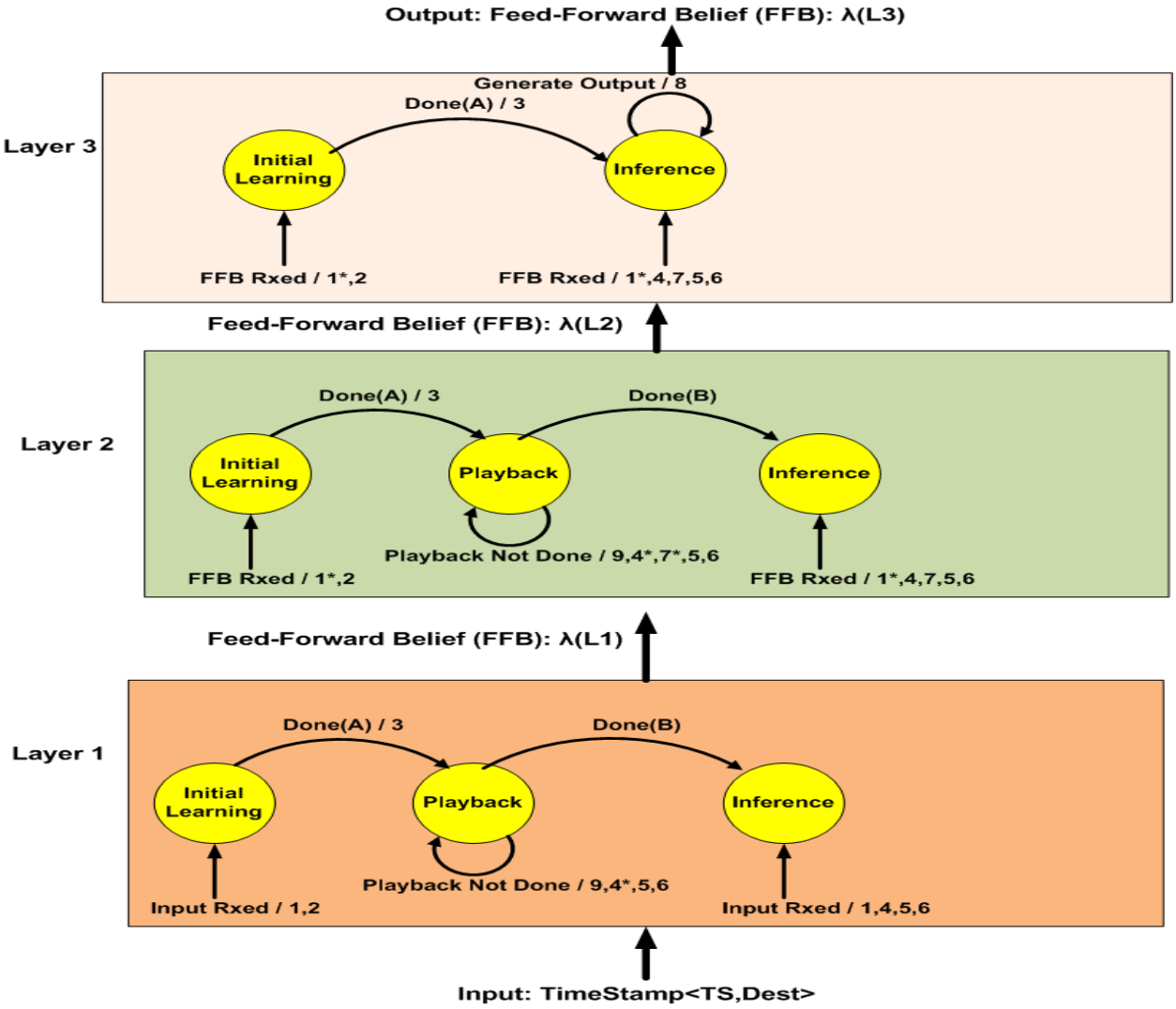


Figure 4 HTM Three layer Implementation

State transitions are shown in the form of **event / set of actions.** Specifically, the following states are defined:

- **Initial Learning** state is entered after the HTM is first created. 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. Learning entails both spatial and temporal learning. Spatial learning at layer 1 covers identification of individual sequences of destinations, while at layer 2 and 3 it covers identification of individual sequences of coincidences (temporal groups' activation levels from the layer below). Temporal learning entails creating a Markov graph which recognizes and can predict combinations of all sequence of destinations or coincidences learned at each HTM layer. Initial learning is completed with creation of temporal groups (Markov Chains) from the Markov graph. These clusters represent destinations or coincidences that are highly temporally correlated based the specific order in which they follow each other.
- **Playback** state is entered when an HTM at layer $\mathbf{L_n}$ completes learning and is used to bootstrap learning for the layer above $\mathbf{L_{n+1}}$ using the already learned sequences at layer $\mathbf{L_n}$. Playback 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.
- **Inference** state is entered at a given HTM layer when that layer completes training. The inference phase covers computation of the feed forward beliefs which define the degree of membership of the input at a specific layer against the sequence of patterns learned at that same layer adjusted for how rare or frequent that input is.

The HTM prototype leverages variable order Markov chains to represent learned sequences at each one of the three layers of the prototype. Input received at each layer, is matched against learned sequences to find the most persistence longest common subsequence learned by this layer of the HTM. Multiple algorithms are implemented by the HTM to provide different ways to measure the similarity between the input received and the learned longest common subsequence that best matches this input. The inference algorithms implemented follow into two main categories: pattern matching and probability based.

Pattern Matching algorithms are based on the following similarity formula (which measures feed forward beliefs -FFB) applied at each layer of the HTM for a given input sequence:

HTM Layers	Pattern Matching Similarity Formulas for FFB
1	FFB_1 = Sequence Similarity + Sequence Persistence
2	FFB_2 = (Sequence Similarity + Sequence Persistence) * Input Activation Level
3	FFB_3= (Sequence Similarity + Sequence Persistence) * Input Activation Level

Table 1 Degree of Similarity/Membership Formulas

Sequence Similarity = (LLCS * weight1) + (LLCSu * weight2)
Sequence Persistence = (Persistence * weight3) , where weight1 + weight2 + weight3 = 1.0,
LLCS = Length of the longest common subsequence computed between the input and all learned sequences at this layer of the HTM divided by the maximum length between the input sequence and the length of longest common subsequence
LLCSu = Length of the longest common substring between the input and all learned sequences at this layer of the HTM divided by the maximum length between the input sequence and the length of longest common substring
Persistence = Number of occurrences of the Longest Common Subsequence matching the input divided by the number of learned sequences in the Markov graph for that HTM layer
Input Activation Level = Average of all Feed Forward beliefs received at a given layer for each input element that makes up the input sequence

The longest common subsequence between the input sequence and HTM learned sequences at each layer is defined as the longest sequence of characters that appear left-to-right (but not necessarily in a contiguous block) in both input and learned sequences. Because the longest common subsequence is not always unique among learned sequences, the algorithm selects always the one with the highest persistence. The longest common substring accounts for consecutive substrings (substrings are consecutive parts of a string, while subsequences need not be) thus allowing to recognize as more similar, sequences of destinations that directly follow each other. So sequences: 8205 and 4820 are more similar since they share a substring (820) than sequences 8205 and 8125 even though they have the same length for longest common subsequence (825).

During inference, the length of the longest common subsequence (needed to compute the degree of similarity of input sequences against each sequence that can be generated by a Markov chain within an HTM layer) is computed using bottom up dynamic programming. The iterative algorithm to compute longest common subsequence length is shown in Figure 5 with an example of its use in Table 2. The longest common substring algorithm is not shown as it is standard procedure in the literature.

```
ComputeLCSL(input, learned-seq)
    - Create a 2 dimensional table "L" with |input| + 1 rows and |learned-seq| + 1 columns
    - Initialize row = |input| + 1 and column = |learned-seq| + 1 to all zeros
    // Compute LCSL
    - m = |input|
    - n =  |learned-seq|
    - For (i=m;  i >=0;  i--)
    - {
    -   For(j=n; j>=0;  j--){
    -     IF(input[i] == '\0' || learned-seq[j] == '\0')
    -           L[i,j]  = 0;
    -     ELSE IF (input[i] == learned-seq[j])
    -           L[i,j]  = 1 + L[i+1, j+1]  ;
    -      ELSE
    -          L[i,j]  = max (L[i+1, j], L[i, j+1]);
    -      ENDIF
    -   }
    - }
    - Return( L[0,0] );   // Result
```

Figure 5 Algorithm to Compute Length of Longest Common Subsequence

As an example, consider computing the LLCS for input $S_1S_2S_1S_6S_4S_3$, against learned sequence $S_2S_1S_3S_2S_1S_4$ which produces an LCS length of 4 as shown in Table 2. The LCS sequence itself can be retrieved by working forwards through table "L".

	S_2	S_1	S_3	S_2	S_1	S_4	
S_1	4	4	3	3	2	1	0
S_2	3	3	3	3	2	1	0
S_1	2	2	2	2	2	1	0
S_6	1	1	1	1	1	1	0
S_4	1	1	1	1	1	1	0
S_3	1	1	1	0	0	0	0
	0	0	0	0	0	0	0

Table 2 Table L used to generate Length of Longest Common Subsequence

In the context of the HTM, specific pattern matching algorithms differ in how they combine feed forward beliefs belonging to a give observation. The Average method simply

averages feed forward beliefs for a given user over a given observation. The Weighted Average Method averages the feed forward beliefs belonging to a give observation with respect to the proportion of input matched that far. The Weighted Average Method computes the proportion of the input matched either a layer 1 of the HTM (BottomUp) or at layer 3 of the HTM (TopTop).

Probability algorithms are based on computation of the path probability (feed forward beliefs at each layer of the HTM) of the input against the learned longest common subsequence that best matches the input. Specifically, the path probability of the input is computed based on the path probability of the learned longest common subsequence in the Markov chain that best matches the input, with adjustments (penalties) made for mismatches against the input sequence. The algorithm that computes the path probability of the longest common subsequence that best matches the input is shown in Figure 6. In Figure 7 the path probability of the longest common subsequence is adjusted based on how well this sequence matches the input sequence.

Computing Path probability of the input at each HTM layer entails two steps:

1: Learned_LCS_Nodes = ComputeLCSNodesProbability(input)

2: ComputePathProbability(input, Learned_LCS_Nodes);

ComputeLCSNodesProbability(input)

// For each layer of HTM, compute the path probability from the Markov graph
// of the learned longest common subsequence (LCS) that best matches the input as follows

Learned_LCS = FindBestMatchingLCSFromMG(input)

For each node V_i (representing node i of the learned LCS) that precedes node V_k (representing node k of the LCS, $V_i \rightarrow V_k$) Do

- Learned_LCS[i].probability = $P(V_k \mid V_i) = V_i V_k \ / \ V_i$

EnDo

Where V_i = Total frequency count of all nodes terminating into node V_i

$V_i V_k$ = Frequency count for transitions from V_i to V_k ($V_i \rightarrow V_k$)

Figure 6 Algorithm to Compute Path Probability – Part 1

```
ComputePathProbability(input, Learned_LCS_Nodes)
// Compute the path probability of the input from the learned LCS applying appropriate penalties for
//  mismatches as follows:
        Path_prob = 1.0
        For each element "e" of the input Do
         IF match is found between "e" and learned_LCS_Nodes[i] at the same relative position "i"
         THEN
                -    Path_prob = Path_prob * learned_LCS_Nodes[i].probability

        Else IF "e" does not match learned_LCS_Nodes[i] OR "e" matches an already matched element of
        learned_LCS_Nodes
        THEN
                // Penalize this input element
                Path_prob = Path_prob * PENALTY

        ELSE IF a match is found between "e" and learned_LCS_Nodes[j] not at the next relative position
        THEN
                // Elements exist in the learned LCS at a position "j" beyond elements at position "i"
                // (last matched element) in the learned LCS that are not part of the input ➔ Penalize them

                -    Path_prob = Path_prob * learned_LCS_Nodes[i].probability * (j - i ) * PENALTY
        EndIF
```

Figure 7 Algorithm to Compute Path Probability – Part 2

All HTMs at layer 3 produce feed forward beliefs (FFBs) during the inference phase that are input to a maximization layer that computes the best FFB among all HTMs based on the configured pattern matching or probability based algorithms previously described.

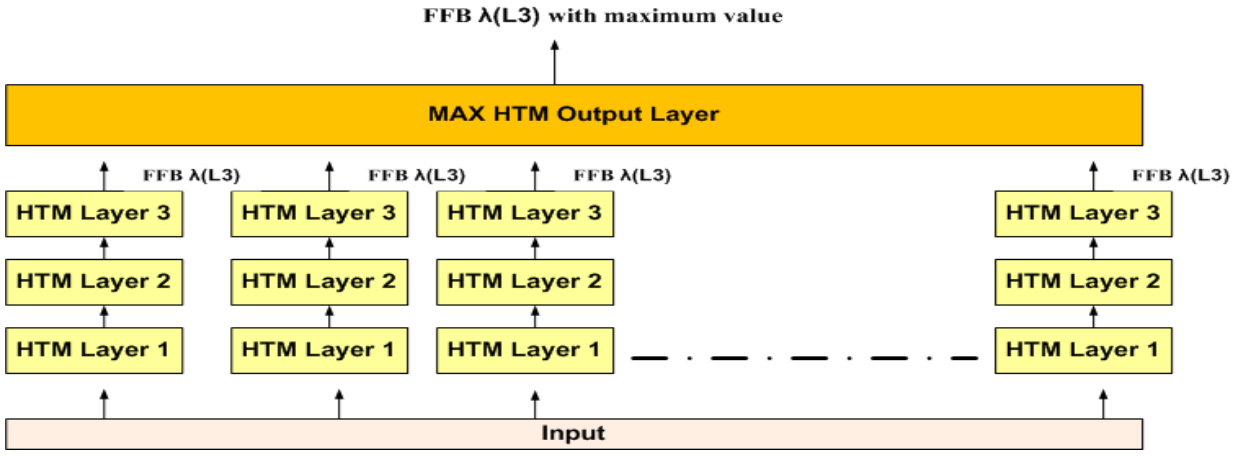


Figure 8 HTM MAX layer used during the inference phase

The tables below show a quick summary of the events and actions associated with the HTM in Figure 4.

Events	Description
Done(A)	Generated when initial learning completes. This occurs when a preconfigured number of observations has been processed.
Done(B)	Generated when all learned sequences in the MG have been generated to the upper layer
Generate Output	Layer 3 outputs results of inference. Note that if no match occurs then nothing is output.

Table 3 HTM State Machine Events

Actions	Description
1	**Spatial Pooler** creates sequences of destinations from individual input destinations
1*	**Spatial Pooler** creates sequences of coincidences from individual instances of FFBs
2	**Temporal Pooler** creates and updates the Markov graph (MG) based on received input (sequences of destinations for layer 1 and sequences of coincidences for layers 2,3)
3	**Temporal Pooler** creates Markov chains (temporal groups) extracted from the Markov graph
4	Find the longest common subsequence that best matches the input sequence. Then: • [Pattern Matching] Compute the degree of similarity of the input sequence based on the specific inference algorithm (Table 1) • [Path Probability] Compute the path probability of the input against the longest common subsequence
4*	In the Playback state, because sequences are internally generated, the degree of similarity of the input is always 100%
5	[Pattern Matching only] Adjust the degree of similarity of input (destination or coincidence) to learned (historical) persistence of longest common subsequence matching that input by computing the LCS persistence (Table 1).
6	Compute feed forward belief (FFB) and send it to higher layer
7	Compute the level of activation of coincidence for layer L_n from FFB from layer L_{n-1} (Table 1). Then: • [Pattern Matching] Adjust degree of similarity with level of activation • [Path Probability] Adjust path probability with level of activation
7*	In the Playback state of layers 2 and 3 the level of activation of input is always assumed to be 100% because the HTM learns the structure of co-occurrences of temporal groups from layers below.
8	Report output in the form of feed forward belief
9	Generate, in time-order, all sequences belonging to each Markov chain (temporal groups) at this layer of the HTM.

Table 4 HTM State Machine Actions

The rest of the section describes in more detail different areas of the HTM implementation. The spatial pooler determines the demarcation point for combining time stamped input in the form of < TCP Timestamp Value, Navigational Destination> into sequences of destinations using the following rules:

a. Arrival of input destination to spatial pooler falls within a specified inter arrival time
b. Input destination is not already present in the sequence (no duplicate allowed in sequence)
c. Size of sequence does not exceed a specified maximum sequence size

For instance, assuming a max sequence size of 5 with a max allowed inter arrival time of 3 *ms*, the following input 1<TS, S1>, 3<TS, S2>, 4<TS, S3>, 5<TS, S4>, 10<TS, S1>, 12<TS, S2>, 13<TS, S5> would be converted by the spatial pooler into the following sequences: S1, S2, S3, S4 and S1,S2,S5. For the rest of the discussion, for illustrative purposes, assume that the following sequences of destinations were formed by the spatial pooler based on input collected from the network, using the rules presented above.

Example Data	Description
S1,S2,S3,S4	User visits four web destinations which, for the purpose of this example, relate to category “S” for “soccer”.
S1,S2,S5	
S1,S3,S6	
S1,S3,S6	
T1,T2,T3	User visits three web destinations which, for the purpose of the example, relate to category “T” for “tennis”.
T1,T3,T5	
T6,T5,T7	
S3,S7,S6,S1	
H-L1,H-L2	User visits two web destinations, which for the purpose of the example, relate to category “H-L” for “High

Example Data	Description
	School".
H-L1	
H-L1,H-L3	
H-L1	
H-L1, H-L2, H-L3	
UL1, UL2, UL3, UL4	User visits four web destinations, which for the purpose of the example, relate to category "UL" for "University".
S2,S6,S5	

Table 5 HTM Sample Input

The temporal pooler creates and updates the Markov graph (MG) for each layer. The MG represents the long term memory for each layer. The MG stores representations of sequences as they occur over time with nodes defining the elements of a sequence and arcs between nodes defining the number of times a node Y follows a node X (X → Y). The MG includes both "start" and "final" state nodes for all sequences represented. More formally, the Markov graph is a Markov model which is characterized by a set of states $\{s_1, s_2, s_3, \ldots s_n\}$ and a transition probability matrix $[Pr_{i,j}]_{nxn}$ where $Pr_{i,j}$ represents the probability of a transition from state s_i to s_j. The probability of reaching state s_j from state s_i is given by the product of all transition probabilities along the non-cyclic path.

In general, Markov models predict a symbol using some finite number (which determines the order of the Markov model) of immediately preceding symbols (history), which is called the Markov context. Variable order Markov models always attempt to identify the longest Markov context possible. While Markov models are probabilistic models for sequences, their predictive power lies in their ability to accurately recognize the Markov context. Markov Chains can do this very well because at their core they are a special form of finite state machines. More formally, a *deterministic finite automaton*, DFA, is a five-tuple $M = (K, \Sigma, \delta, s, F)$ where K is a finite set of

states $K=\{k_1,\ldots k_n\}$, Σ is a finite input alphabet $\Sigma=\{ a_1,\ldots a_n \}$, $s \in K$ is the initial state, $F \subseteq K$ is the set of final states, and δ is the transition function mapping $K \times \Sigma$ to K where $\delta (k_i,a)$ that represents the state reached when in state k_i and the input symbol *a* is read. A *probabilistic finite automaton* (PFA) is a finite automaton that has a probability attached to each transition between states. A PFA having at most one transition between every two states corresponds to a Markov chain (Borges, 2000).

In this study, the Markov graph can be thought of as represented by a two dimensional matrix with rows holding the "start" state and all individual elements of learned sequences. Columns hold, excluding the "start" state, the same individual elements of learned sequences, followed by the "final" state. The temporal pooler updates the MG by adding new rows/columns for each new (not seen before) element of a sequence. The temporal pooler also increments the transition frequency counts for links between existing elements of a sequence. An actual implementation of Markov graphs would not have been able to use a two dimensional matrix as just described since the Markov graphs would potentially need to learn and grow continuously, instead the Markov graph was implemented using an adjacency list with *n* vertices consisting of *n* lists. The i^{th} list would have a node for vertex *j* if and only if the graph contains an edge from vertex *i* to vertex *j*. This node would contain the relevant values for vertex *j* (creation time stamp and frequency count) .

Accuracy of Markov Chains and State Cloning

State cloning was first introduced by Cormack and Horspool (1987) to enable Markov graphs to better discover correct correlation between states. State cloning is needed to prevent

the Markov graph and Markov Chains from either generating or from recognizing incorrect sequences (sequences that had never been learned). Ambiguities occur in these graphs when multiple sequences pass through shared states, loops are an example of such condition. Consider the following Markov Graph which was created with sequences *abd* and *xbc*:

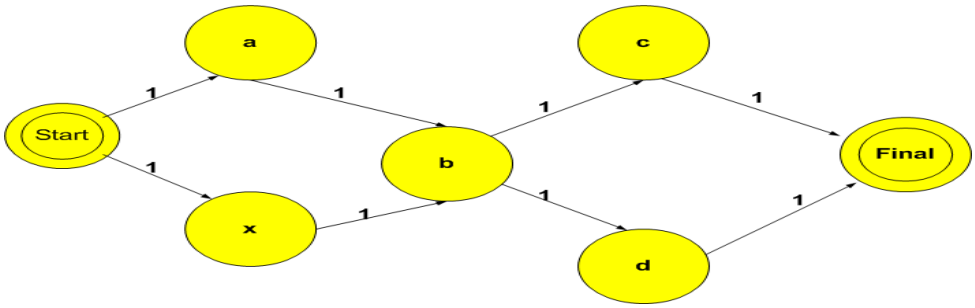


Figure 9 The need for State Cloning

Without knowledge of the input, this graph will recognize and generate one of the following four sequences: *abd*, *abc*, *xdb* or *xbc*. Two of them, *abc* and *xdb* were never added to the graph and are thus incorrect. Assuming that sequence *abd* was added first to the Markov Graph and assuming the following single node clone conditions: Clone a state ***Sx*** (not a "start" or "final" state) iff both conditions are met:

1. Number of out links leaving $\boldsymbol{Sx, O_{sx} > 1}$
2. Number of in links entering $\boldsymbol{Sx, I_{sx} > 1}$

Then cloning node "b" produces the following Markov graph:

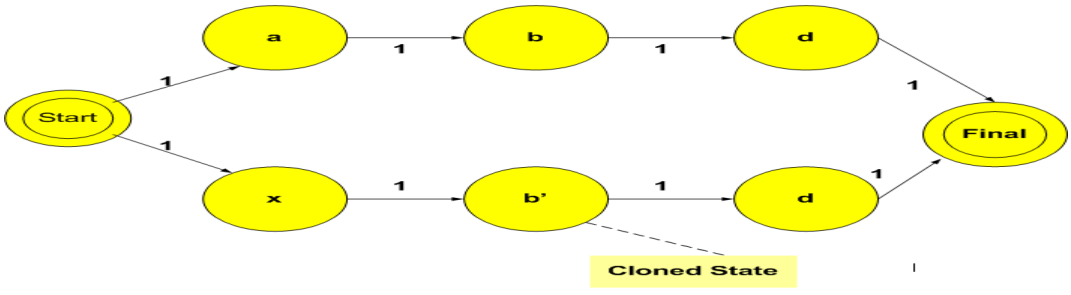


Figure 10 An example of State Cloning at the single node level

While the process of cloning states helps address the ambiguity problem described, it does add more nodes to the graphs and thus does not tend to scale well. The number of clones decreases as cloning conditions are relaxed (increase upper limit for $\boldsymbol{O_{sx}}$ and $\boldsymbol{I_{sx}}$). Unfortunately, increasing this upper limit will re-introduce the ambiguities discussed.

Consider how the defined single node clone conditions can be used when adding a transaction $\mathbf{T_{k,i,j}}$ between three Markov graph nodes. Assume transaction $\mathbf{T_{k,i}}$ has already been added to the graph, then we have three different possibilities:

A. Adding transition $\mathbf{T_{i,j}}$ to $\text{node}_\mathbf{i}$ could violate the "clone conditions" for $\text{node}_\mathbf{i}$. If the clone condition is violated then $\text{node}_\mathbf{i}$ needs to be cloned as shown in Figure 11 case A.
B. Adding transition $\mathbf{T_{i,j}}$ to cloned $\text{node}_\mathbf{j}$ could violate "clone conditions for $\text{node}_\mathbf{j}$. If the clone condition is violated then $\text{node}_\mathbf{j}$ needs to be cloned as shown in Figure 11 case B.
C. Adding transition $\mathbf{T_{i,j}}$ to cloned $\text{node}_\mathbf{j}$ could violate "clone conditions for both $\text{node}_\mathbf{i}$ and $\text{node}_\mathbf{j}$. If the clone condition is violated then both $\text{node}_\mathbf{i}$ and $\text{node}_\mathbf{j}$ need to be cloned as shown in Figure 11 case C.

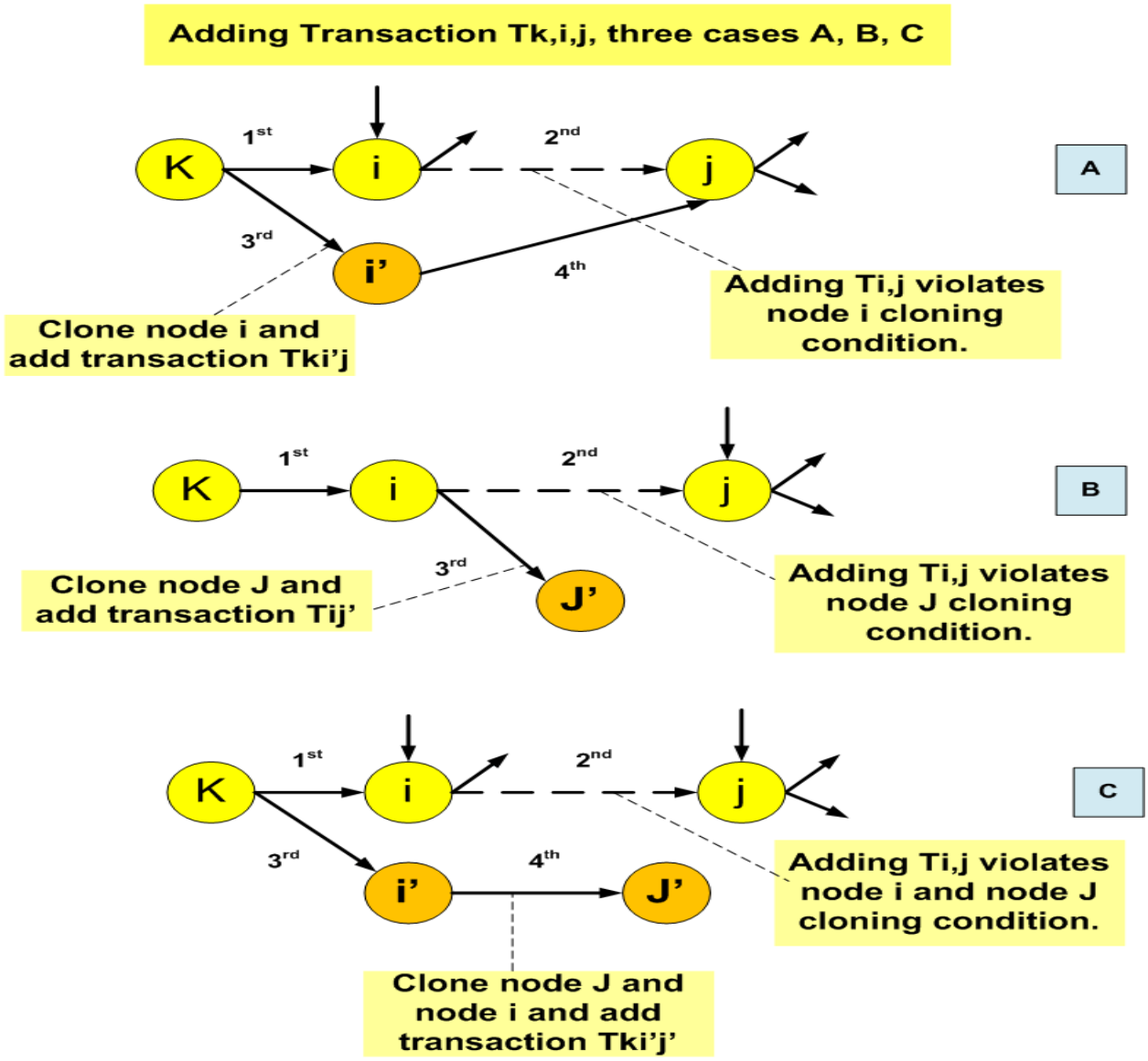


Figure 11 Cloning States in a Markov Graph

Another way to determine when the single node cloning condition is violated is to determine if the number of different sequences that are shared by a node is greater than 2 as shown below.

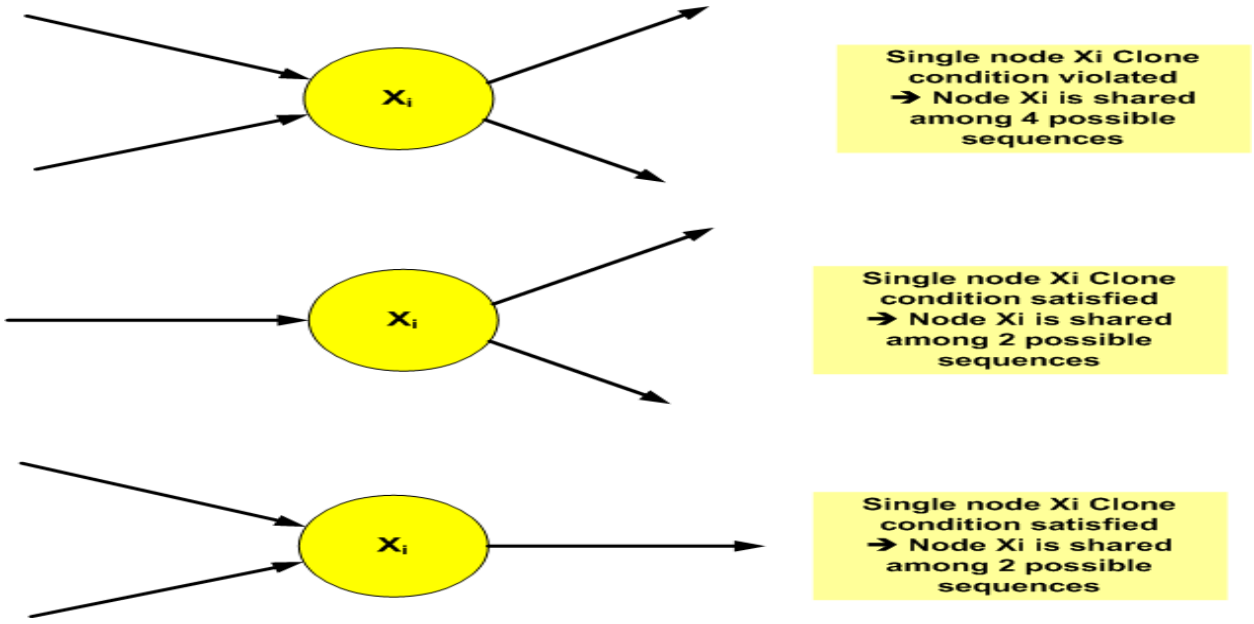


Figure 12 Generalization of Single node cloning

Is it possible to satisfy the single node clone condition and still produce ambiguity within a Markov graph? Yes, consider the Markov graph below where two sequences are added, 5,1,2,3 and 1,2,3,4 in that order. Note that the single node cloning conditions are satisfied, yet this graph produces two sequences that were never learned: 1, 2, 3 and 5, 1, 2, 3, 4.

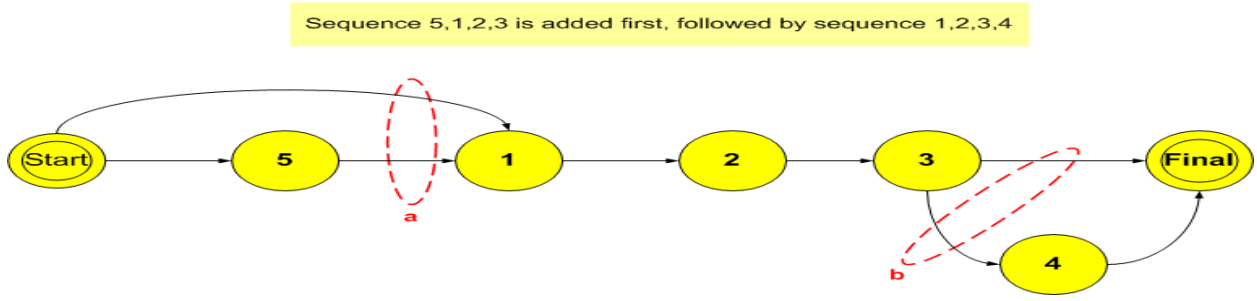


Figure 13 How Single Node Cloning Falls Short

The problem occurs at the transitions covered by points *a* and *b*. These transitions allow the generation through 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 single node cloning condition must be extended to cover multiple nodes in a sequence as shown below.

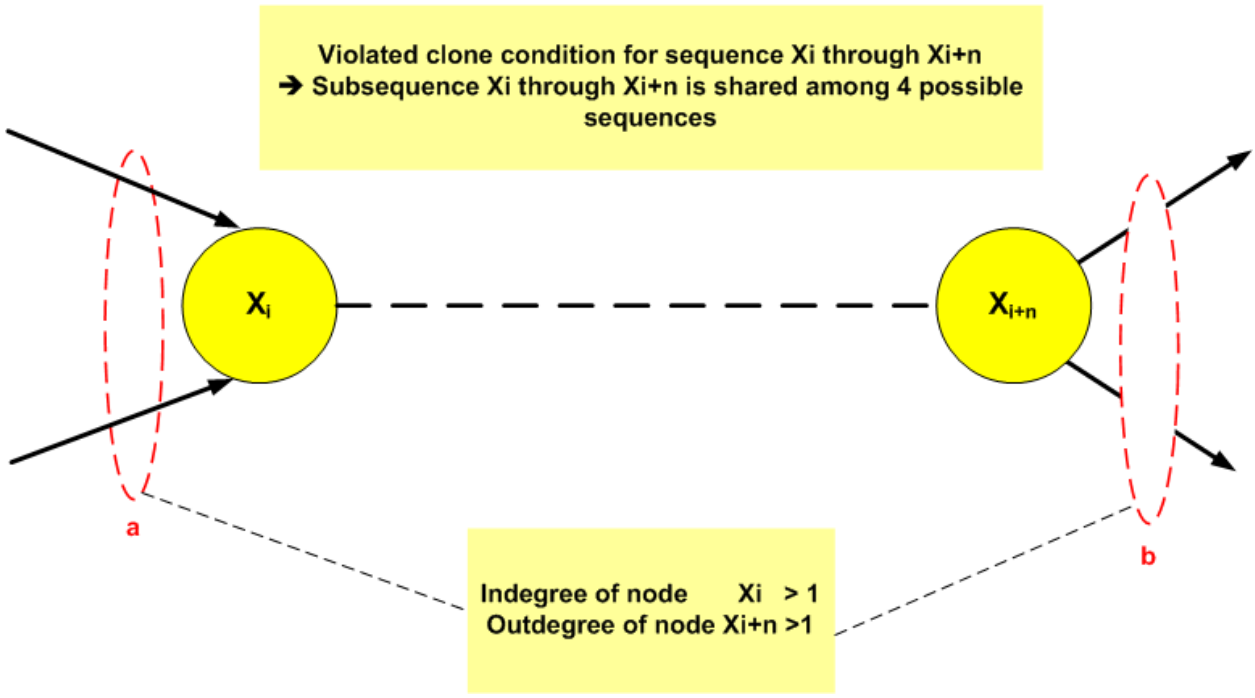


Figure 14 Generalization of Sequence Cloning Condition

The single node clone condition can then be replaced by the following sequence cloning condition. The sequence cloning condition is violated iff:

When adding a sequence to the Markov graph if the in-degree of a node(x) corresponding to sequence element x is going to be > 1 and the out-degree of node(y) (where y ≥ x, that is y can be the same node as x or follow x) corresponding to element y is also going to be > 1 then one must clone all sequence elements corresponding to nodes between node(x) and node(y) that already exist in the Markov graph. Note that sequence cloning is a special case of single node cloning where node(x) and node(y) are one and the same. Figure 15 shows how sequence cloning can be applied to remedy the shortcoming of single node cloning.

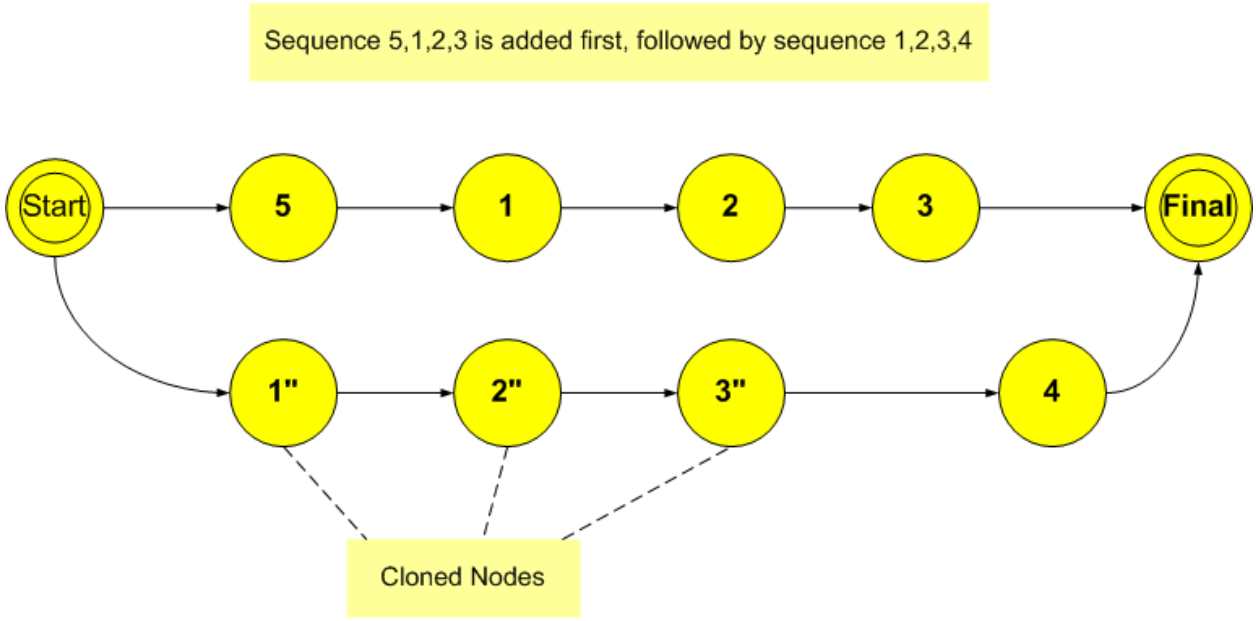


Figure 15 Ensuring that the Sequence Cloning Condition is met

Research on State Cloning

In Hawkins et al. (2009) the authors acknowledge that in order to mimic the operation of a sequence memory in the neocortext, a memory mechanism is needed that can learn and represent sequences of arbitrary high order. These authors recognize that the amount of memory required to keep track of dependencies in long sequences grows exponentially with the order of the model. For this reason they propose to use variable order Markov models that can learn long and complex sequences with manageable amount of resources. Hawkins et al. (2009) proposed a state splitting (cloning) algorithm based on the work of Cormack and Horspool (1987), in order to address the problem that Markov graphs can misrepresent learned sequences when a given state in a Markov graph participates in more than one unique sequence. Based on this algorithm, a state *t* is split (cloned) when it frequently follows a particular state and it follows other states as well. More formally state *t* is split in two states if the following two conditions exist:

$\mathbf{T_{s,t} \geq min\_cnt1,}$ where $\mathbf{T_{s,t}}$ is the transition from state *s* to state *t*

$\mathbf{\Sigma_{i,j \neq s}\ T_{i,t} \geq min\_cnt2,\ min\_cnt1}$ and **min_cnt2** are fixed threshold values

The concept of state cloning in the context of usage mining was first introduced in Levene and Loizou (2003). Based on the state cloning algorithm defined by these authors, given a transition $T_{i,k,j}$, a state s_k is cloned if there is sufficient evidence that the transition from s_k to s_j is dependent on the transition from state s_i to s_k. More formally:

$$\mathbf{P_{i,k,j} - \frac{1}{Lk} > \gamma_k}, \quad \mathbf{P_{i,k,j} = w_{i,k,j} / w_{k,j}}$$

Where $w_{i,k,j}$ is the frequency count from s_k to s_j given that the previous transition that occurred was from s_i to s_k and $0 < \gamma_k < 1$. Borges and Levene (2004) proposed a cloning algorithm where a state would be cloned when the second order probability differs from the first order probability by more than a given threshold based on the following four conditions. A state A_x is eligible for cloning iff:

1. State *s* has at least two out-links O > 1
2. State *s* has at least two in-links I > 1
3. Wx > V, where V represents the number of visits (Wx) to a state to ensure the reliability of the probabilities associated with the state
4. In the context of transition A_{jxk}, there is at least one transition (A_j,A_x) and (A_x,A_k) such that $\mathbf{|P_{j,x,k} - P_{x,k}| \geq \gamma}$, where $0 \leq \gamma \leq 1$, $\mathbf{P_{j,x,k} = w_{j,x,k} / w_{j,x}}$ and $\mathbf{w_{j,x}}$ is the number of times that the link from A_j to A_x was traversed, and $\mathbf{w_{j,x,k}}$ is the number of times that sequence $A_jA_x\ A_k$ was traversed.

The ability of Markov graphs to accurately represent unique learned sequences decreases when a state in the Markov graph is shared among more than one sequence. All cloning approaches described so far rely on a threshold to mitigate the Markov graph accuracy problem by measuring the number of occurrences of certain transitions coming to or leaving certain states. Cormack and Horspool (1987) and Borges and Levene (2004) place more importance on transitions leading up to the target state (state that can be cloned), while Levene and Loizou (2003) put more importance on transactions that follow the target state. The key idea of using a threshold that measures the frequency of state transitions is to ensure that if a transition that creates ambiguities within a Markov graph occurs with enough persistence then it is worth cloning that target state. In contrast, the "sequence cloning" algorithm proposed in this paper targets exclusively the structure of the Markov graph and can operate across multiple target states that make up a sequence by always assuming that any transition in a Markov graph that creates ambiguities will trigger cloning of one or more target states. As a result, none of the described cloning approaches would be able to address the ambiguity shown in Figure 13, but the "sequence cloning" approach proposed in this paper does.

Playback and Inference

While the ability to recognize sequences is critical during the inference stage, sequence generation is critical in both playback and inference stages. 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 queue timestamps associated with nodes with the least recent (oldest) timestamp for each transition up to the "final" state.

Generation of sequences stored in Markov chains is also performed during inference when the input sequence produced by the spatial pooler is compared for degree of similarity against all learned sequences that can be produced by Markov Chains at that layer. In this case sequence generation does not need to be in time order and thus it ignores timestamps held at each Markov chain node.

The temporal pooler creates Markov Chains from the Markov graph (MG). Markov chains are also Markov Models which represent clusters of highly connected sequences. Markov Chains were implemented as overlays of the Markov graph. Figure 16 shows a Markov graph with 6 Markov Chains (g1-g6) based on the sample data input from Table 5. Clone states were created as each sequence was added to the Markov graph and links between nodes represent the frequency of occurrences of transitions between those two nodes.

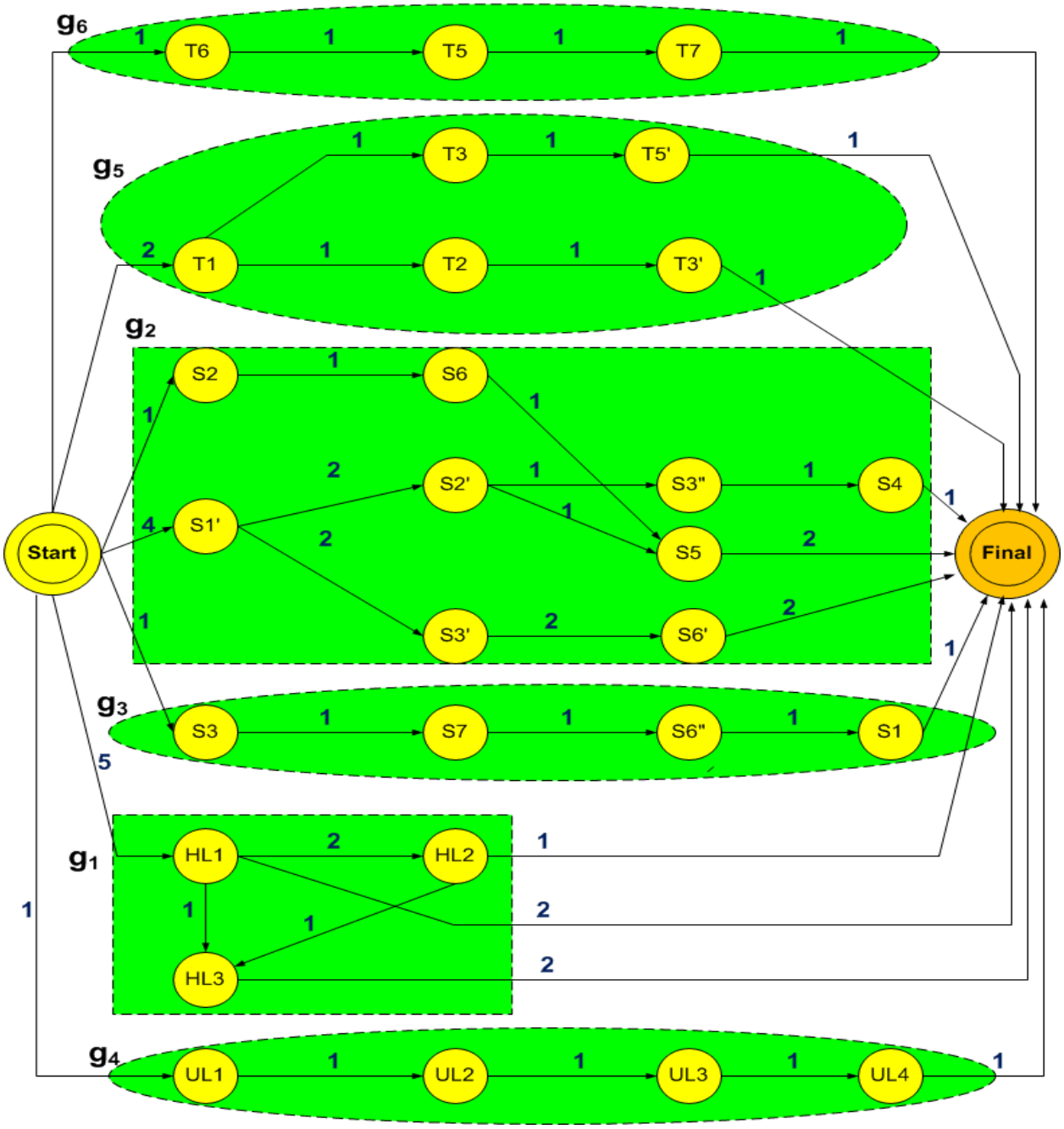


Figure 16 Layer 1 Markov Graph and Markov Chains (Temporal Groups g_1-g_6)

The algorithm used to create the Markov chains shown in Figure 16 is presented below

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 17 Markov Chains Creation Algorithm

The algorithm shown in Figure 17 guarantees that highly connected nodes that form a complete sequence belong to the same Markov chain and that each node (cloned and equivalent non cloned nodes are considered to be different unique nodes in a Markov graph) belongs to one and only one Markov chain. The merge portion of the algorithm ensures that nodes S2 and S6 belong to Markov chain g2 since node S5 also belongs to g2. The agglomerative hierarchical clustering algorithm used in George and Widrow (2008) could not be used in this study since it relies on grouping based on measuring similarity which, in the context of image pixels, makes sense but when used with web destinations, has no meaning. George and Jaros (2007) propose a simple algorithm to create Markov chains from a Markov graph. This algorithm creates Markov chains based on the degree of connectivity of nodes, so that the most highly connected nodes are grouped together. In particular, the algorithm finds the next seed as the most connected node (has the highest aggregated in-degree frequency value) and groups it with the next N_{top} (fixed value)

nodes (not already in the group) that are most connected (have the highest frequency values) to the seed node. The algorithm in Figure 17 accomplishes a similar goal but instead of relying on the strength of the connectivity and using an arbitrary value for N_{top}, it completely relies on the existence of connectivity among the nodes to each other ignoring the strength of the connections. In doing so, algorithm in Figure 17 manages to preserve the integrity of sequences learned regardless of the number of nodes in the graph and degree of connectivity among these nodes.

Walking Through an Example

In order to get a better idea of how beliefs propagate up the HTM network layers, the rest 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 Figure 16. 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 6.

Sequence Generated	**Historical Persistence**	**Feed Forward Belief λ<g1,g2,g3,g4,g5,g6>**
S1,S2,S3,S4	W_{1234}/total # seq in g_2 = 1/15= 0.67	λ<0,.67,0,0,0,0>
S1,S2,S5	W_{125}/total # seq in g_2 = 1/15= 0.67	λ<0,.67,0,0,0,0>
S1,S3,S6	W_{136}/total # seq in g_2 = 2/15= 0.13	λ<0,.13,0,0,0,0>
S1,S3,S6	W_{136}/total # seq in g_2 = 2/15= 0.13	λ<0,.13,0,0,0,0>
T1,T2,T3	W_{123}/total # seq in g_5 = 1/15= 0.67	λ<0,0,0,0,.67,0>
T1,T3,T5	W_{135}/total # seq in g_5 = 1/15= 0.67	λ<0,0,0,0,.67,0>
T6,T5,T7	W_{657}/total # seq in g_6 = 1/15= 0.67	λ<0,0,0,0,0,67>
S3,S7,S6,S1	W_{3761}/total # seq in g_3 = 1/15= 0.67	λ<0,0,67,0,0,0>
H-L1,H-L2	W_{12}/total # seq in g_1 = 2/15= 0.13	λ<.13,0,0,0,0,0>
H-L1	W_1/total # seq in g_1 = 5/15= 0.33	λ<.33,0,0,0,0,0>
H-L1,H-L3	W_{13}/total # seq in g_1 = 1/15= 0.67	λ<.67,0,0,0,0,0>
H-L1	W_1/total # seq in g_1 = 5/15= 0.33	λ<.33,0,0,0,0,0>
H-L1, H-L2, H-L3	W_{123}/total # seq in g_1 = 1/15= 0.67	λ<.67,0,0,0,0,0>
UL1, UL2, UL3, UL4	W_{1234}/total # seq in g_4 = 1/15= 0.67	λ<0,0,0,.67,0,0>
S2,S6,S5	W_{265}/total # seq in g_2= 1/15= 0.67	λ<0,.67,0,0,0,0>

Table 6 Feed Forward Beliefs generated at Layer 1 during Playback

After layer 1 completes initial training, layer 1 starts playback of learned sequences to layer 2. The spatial pooler at layer 2 maps feed forward beliefs from layer 1 into sequence of coincidences using the rules previously described to combine input into sequences as shown in Table 7.

Feed Forward Belief	Coincidence	Sequences of Coincidences for Layer2
λ<0,.67,0,0,0,0>	C_1 new coincidence	C_1
λ<0,.67,0,0,0,0>	C_1	C_1
λ<0,.13,0,0,0,0>	C_1	C_1
λ<0,.13,0,0,0,0>	C_1	
λ<0,0,0,0,.67,0>	C_2 new coincidence	C_1, C_2
λ<0,0,0,0,.67,0>	C_2	
λ<0,0,0,0,0,67>	C_3 new coincidence	
λ<0,0,67,0,0,0>	C_4 new coincidence	
λ<.4,0,0,0,0,0>	C_5 new coincidence	C_2, C_3, C_4, C_5
λ<1,0,0,0,0,0>	C_5	C_5
λ<.2,0,0,0,0,0>	C_5	C_5
λ<1,0,0,0,0,0>	C_5	C_5
λ<.2,0,0,0,0,0>	C_5	
λ<0,0,0,1,0,0>	C_6 new coincidence	
λ<0,.2,0,0,0,0>	C_1	C_5, C_6, C_1

Table 7 Layer 2 Spatial Pooler conversion of layer 1 temporal groups into layer 2 sequence of coincidences

Having completed initial learning, layer 2 then would convert the received coincidences into the Markov Graph and Markov chains as shown below in Figure 18.

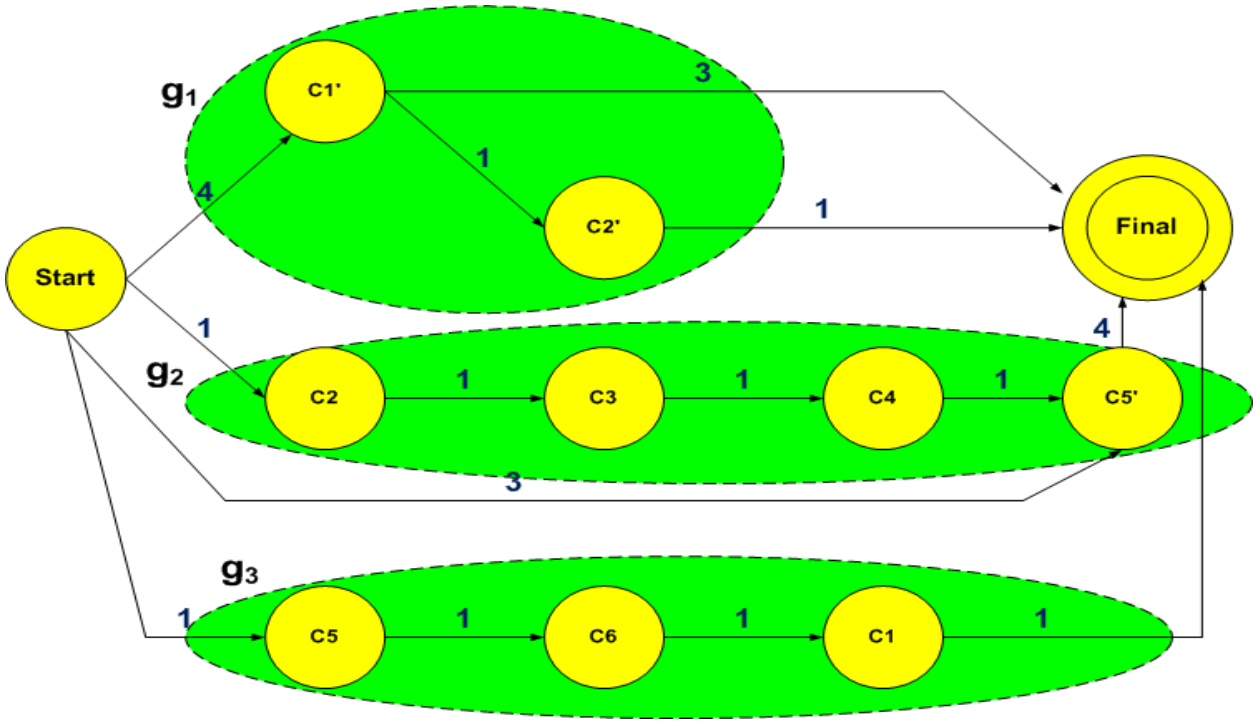


Figure 18 Layer 2 Markov Graph and Markov Chains

Assuming that initial learning is completed at layer 2, layer 2 starts playback in order to train layer 3 as shown in Table 8.

Sequence Generated	Historical Persistence	Feed Forward Belief λ<g1,g2,g3>
C_1	W_1/total # seq in g_1 = 4/9= .44	λ<.44,0,0>
C_1	W_1/total # seq in g_1 = 4/9= .44	λ<.44,0,0>
C_1	W_1/total # seq in g_1 = 4/9= .44	λ<.44,0,0>
C_1 C_2	W_{12}/total # seq in g_1 = 1/9= .11	λ<.11,0,0>
C_2, C_3, C_4, C_5	W_{2345}/total # seq in g_2 = 1/9= .11	λ<0,.11,0>
C_5	W_5/total # seq in g_3= 1/9= .11	λ<0,.11,0>
C_5	W_5/total # seq in g_3= 1/9= .11	λ<0,.11,0>
C_5	W_5/total # seq in g_3= 1/9= .11	λ<0,.11,0>
C_5, C_6, C_1	W_{561}/total # seq in g_3 = 1/9= .11	λ<0,0,.11>

Table 8 Feed Forward Beliefs generated at Layer 2 during Playback

Finally, the spatial pooler at layer 3 converts feed forward beliefs received from layer 2 into sequence of coincidences using the rules previously described to combine coincidences into sequences of coincidences as shown in Table 9.

Feed Forward Belief	Coincidence	Sequences of Coincidences for Layer3
λ<.44,0,0>	C_1 new coincidence	C_1
λ<.44,0,0>	C_1	C_1
λ<.44,0,0>	C_1	C_1
λ<.11,0,0>	C_1	
λ<0,.11,0>	C_2 new coincidence	C_1, C_2
λ<0,.11,0>	C_2	C_2
λ<0,.11,0>	C_2	C_2
λ<0,.11,0>	C_2	
λ<0,0,.11>	C_3 new coincidence	C_2, C_3

Table 9 Layer 3 Spatial Pooler conversion of layer 2 coincidences into layer 3 sequence of coincidences

Figure 19 shows how low level user navigational concepts, represented by different navigational patterns, move up the HTM hierarchy and form higher level navigational concepts.

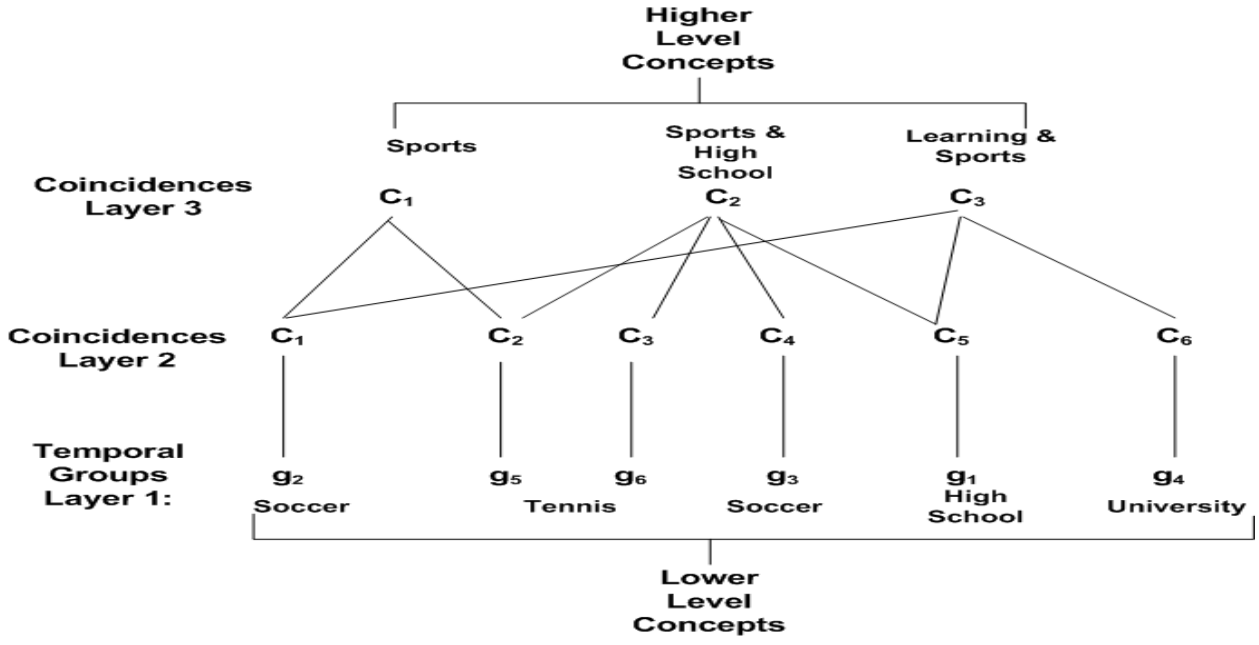


Figure 19 Creation of higher level navigational concepts at higher levels of the HTM

Figure 20 shows the sequential relationships of higher level (layer 3) navigational patterns for this user.

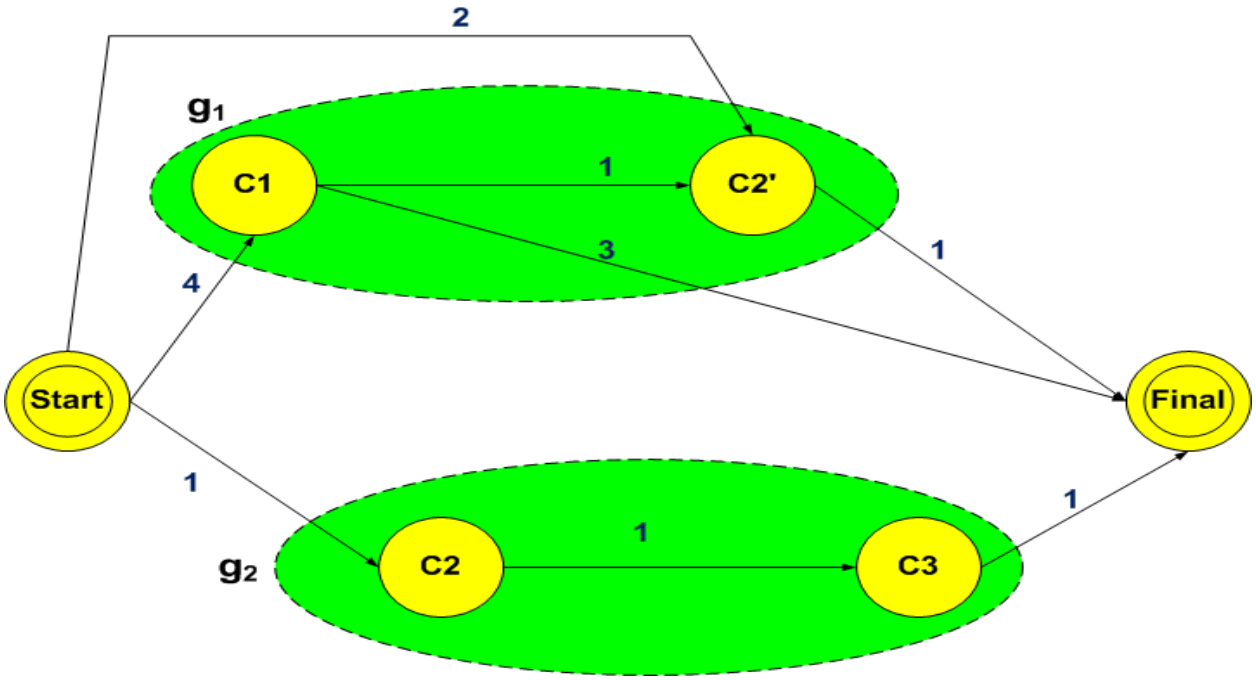


Figure 20 Layer 3 Markov Graph and Markov Chains

Revisiting Markov Chains Accuracy

In Figure 18, consider coincidence node *C5* in temporal group g3 and its cloned equivalent *C5*' in temporal group g2, this node should represent the start of sequences belonging to the same temporal group, g3, but it does not. Single occurrences of C5 will be learned in layer 2 and forwarded to layer 3 as coincidences belonging to temporal group 2 instead of temporal group 3. This observation leads to the requirement that each node in a Markov graph that follows the "start" state must be unique. Figure 21 shows the Layer 2 Markov graph and Markov chains that address the problem.

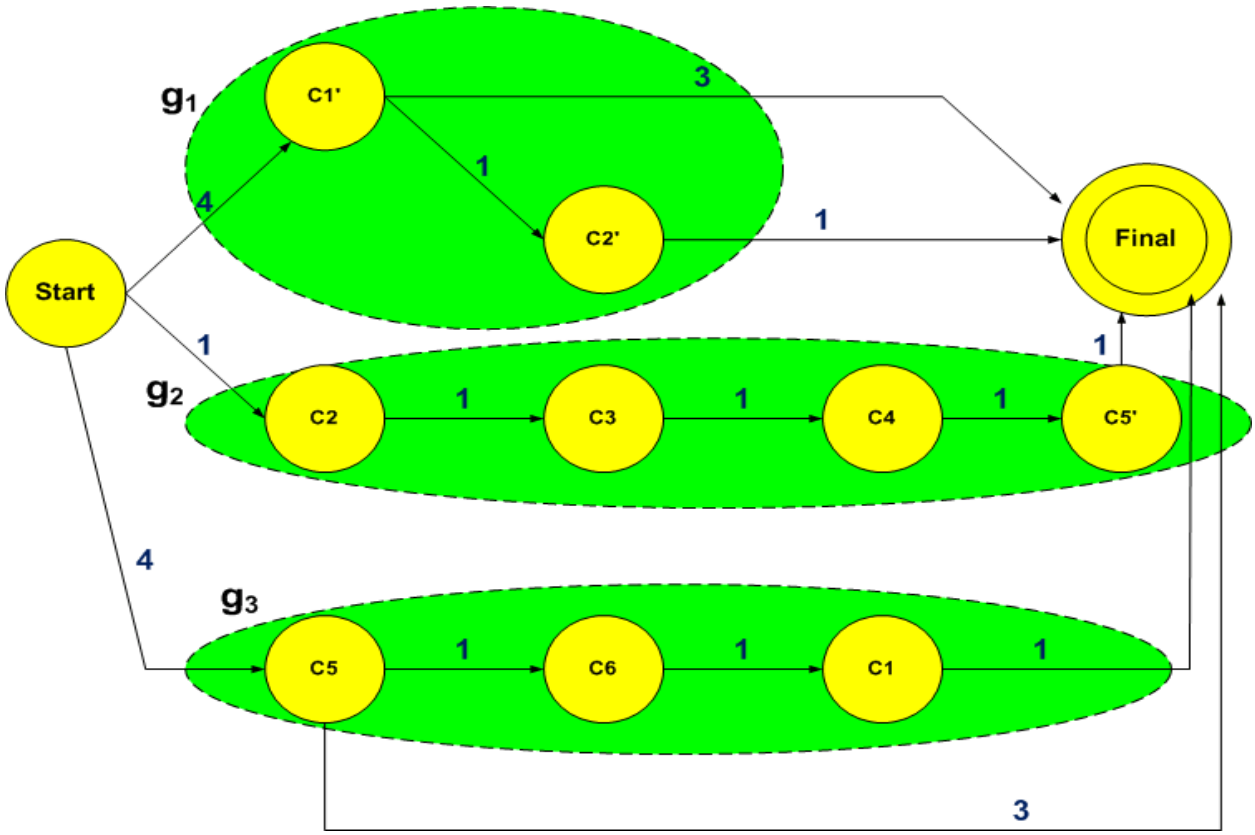


Figure 21 Addressing Layer 2 Markov Chains Ambiguity with node C5

The condition necessary to address this problem is the following:

When a cloned node *N'* representing the start of a an input sequence is also adjacent to the "Start" state (as its equivalent non cloned node) and the out degree of the cloned node *N'* is 1, then the transition to this cloned node *N'* must be moved (updating frequency counts for the transition into and out of both the cloned and non cloned nodes) to the equivalent non cloned node *N* in the Markov graph. As a result, the transition with a frequency of 3 from the "start" state to cloned state C5' shown in Figure 18, is moved in Figure 21 to non cloned state C5.

Alternative Approaches

The objective of this section is to introduce several approaches of which a few were chosen as a basis of comparison against which the HTM approach was measured. The ability to recognize ordered sequences of web visits by users is critical in order to accurately attribute past user communication activity to a specific user. This section explores different techniques used in sequence data mining by leveraging the research in this area conducted by Sarawagi, S. (2005), more recent research can also be found in (Xing, Pei & Keogh, 2010). These approaches can be used with streams of data much like the HTM.

A sequence is an ordered set of elements $s = a_1, a_2, a_3, \ldots, a_n$, where each element a_i could be numerical, or categorical as is the case for a fixed size alphabet Σ. The length of a sequence is not fixed and the order, which can be regular or irregular, is determined by time or position. The need to analyze sequences is evident in multiple areas of research: sequence of phonemes in speech recognition, sequence of words and delimiters in language analysis, sequence of strokes in handwriting, in bioinformatics for gene or protein analysis, in website/ecommerce mining where work has been done on modeling a customer as a sequence of page visits/items orders and using that to classify customers or predict the next page to be visited. There are several operations that can be applied to sequences, from traditional data mining operation like classification, clustering, and discovery of repeated patterns, to sequence specific operations like partial sequence classification, segmenting a sequence, and predicting the next symbol of a sequence.

Sequence classification assumes the existence of a set of classes *C* and a number of example sequences in each class. The model is trained so that unseen sequences can be identified as belonging to a given class. For instance, in intrusion detection, given a sequence of packets, the model would label a session as an intrusion or as normal. Two key characteristics make

sequence classification unique: sequences are of variable lengths and order does matter. Several traditional classifiers can be adapted to be used with sequences: boundary-based, generative, distance-based, kernel-based.

Classification methods such as decision trees, neural networks, linear discriminants are boundary-based classification schemes. These schemes all require data to have a fixed set of attributes in order to map each instance to a point in a multi-dimensional space. During training each class is partitioned into a separate region of the multi-dimensional space, then predicting the class label that the given instance *x* belongs to entails finding the boundaries of the region that *x* belongs to, and mapping it to the associated class determined during training. Several methods have been used to embed sequences in a fixed dimensional space. In text mining for instance, sequences of words (terms) are cast as a vector where each term represents a dimension and its coordinate represents the frequency count (the term and inverse document frequency TF-IDF). The similarity between any two documents is measured using a cosine similarity measure. While this approach is quite effective in finding frequently used terms and reducing the discriminative power of terms that appear frequently in many documents, this approach ignores the order of sequence elements (Han & Kamber 2006). Another approach leverages a sliding window technique to map subsets of sequences to a fixed dimensional space. The approach uses a k-window size to create k dimensions corresponding to *k-grams* of elements of a sequence. The number of dimensions is bound by d^k where *d* is the size of the specific domain. This approach represents an improvement over the vector approach previously described since it is able to keep the order of the elements in the k-gram. Each k-gram represents a segment of *k* consecutive segments and is usually selected as a feature. The sequence is then represented as a vector of the presence or absence of k-grams or as a vector of the frequencies of k-gram. The sliding window

method has been used to classify sequences of system calls as being representative of intrusions or normal application behavior (Lee & Stolfo, 1998).

Generative classifiers require a generative model of the data for each class since they assume that sequences in a class are generated by the underlying model. For each class *i*, the idea is to train a generative model $GM_{i,}$ to maximize the likelihood over all training instances of sequences in class *i.* The prior probability of a class $P(c_i)$ is the fraction of training instances of sequences in class c_i. For predicting the class of an instance *s* of a sequence, one can assume class conditional independence and use naïve Bayesian classification $P(s|c_i)*P(c_i)$ for each *i* and then select the class *i* with the largest value of $P(s|c_i)*P(c_i)$. Generative models differ in how much importance they place on dependence on specific parts of the sequence (the context).

The simplest generative model is the independent model which assumes that the probability distribution of an element at position *i* of a sequence is independent of all elements before it, that is $P(x_i \mid x_1, \ldots x_n) = P(x_i)$. Given a set of training sequences *T*, the probability of a subsequence $s \in T$ is estimated as a fraction of the number of occurrences of s in *T*. In a first order Markov model the probability of generating the i^{th} element depends on the element immediately preceding it. Thus the conditional probability $P(x_i \mid x_1, \ldots x_n) = P(x_i \mid x_{i-1})$ and during training, the maximum value of $P(s_i \mid s_x)$ is estimated as the ratio of $s_x\ s_i$ occurrences in T over the occurrences of s_x. For higher order Markov models the probability of generating an element of a sequence at position *i* depends on a fixed length *r* of symbols before it. Thus the conditional probability $P(x_i \mid x_1, \ldots, x_n) = P(x_i \mid x_{i-r, \ldots,}\ x_{i-1})$ and during training the maximum value of conditional probability $P(s_i \mid s_{xr, \ldots,}\ s_{x1})$ is estimated as the ratio of $s_{xr, \ldots,}\ s_{x1}\ s_i$ occurrences in T over the number of occurrences in $s_{xr, \ldots,}\ s_{x1}$. Markov models have been used in a variety of applications including predicting the user browsing behavior, however since these models do not

utilize enough history they do not correctly discriminate different observed patterns. On the other hand, using higher Markov models has a number of limitations which include high state-space complexity, reduced coverage (number of different sequences recognized) and often even worse prediction accuracy (Deshpande & Karypis, 2004).

One way to address the space and accuracy problem of higher order Markov models is to train different order Markov models and use them all during the inference phase. This approach was proposed by Pitkow and Pirolli, (1999) for an all-K^{th} Markov model. This approach, however sacrifices state-space complexity for improved accuracy. The same authors proposed to identify patterns of frequent access in the form of longest repeating subsequences to produce a subset of all paths in the model (thus removing low information elements from the model) and then use this set of sequences for prediction. This approach does reduce the state-space complexity but also reduces prediction accuracy. Deshpande and Karypis (2004) propose an approach that has low state complexity, improved prediction while retaining the coverage of an all-K^{th} order Markov model. These authors propose three approaches that attempt to eliminate superfluous states, by pruning states of an all-K^{th} order Markov model, while attempting to maintain overall performance. The authors claim that for many problems they can prune up to 90% of the states of an all-K^{th} model with improved accuracy by up to 11%. Their tests show that as the order of Markov model is increased, accuracy tends to increase in line with space requirements (number of states) while coverage decreases. These authors also discovered that by increasing the order of the Markov model the number of states increases which causes the number of training instances needed to train the model to also increase. The three approaches proposed by Deshpande and Karypis (2004) all start from a K^{th} order Markov model and eliminate (prune) many of states in the model that are expected to have low predication

accuracy. In the first approach, a support pruned Markov model is proposed where low support states (with frequency of occurrences below a given threshold) are eliminated without affecting the overall accuracy and coverage of the model. In the second approach, a confidence pruned Markov model is proposed where if the probability of the most frequently traversed out-link of a state is significantly larger than the probability of other out-links emanating from this same state, then this state is kept, otherwise the state is pruned as it is unlikely to yield high accuracy. The probability for this approach is computed based on confidence intervals built around the frequency of state out link traversals. In the third approach, an error pruned Markov model is proposed which computes the error at each state to support a pruning decision. The approach runs the model against a known validation set (a data set not used in training) and then computes errors based on deviations between the results and the known baseline of the validation set.

Variable order Markov (VMM) models attempt to learn probabilistic finite state automata over a finite alphabet, which can model sequential data of considerable complexity. In contrast to N-gram Markov models (0 to N order Markov models) that estimate conditional probabilities of the form $P(\sigma \mid s)$ where s is the context (one or more symbols) and σ is the symbol appearing after the context, Variable order Markov models learn such conditional probabilities where context lengths $|s|$ vary in response to available statistics in the training data. Thus VMMs allow capturing of both small and large order Markov dependencies based on observed data (Begleiter, El-Yaniv & Yona 2004).

There exist a relation between prediction of finite sequences and lossless compression algorithms where in theory, any lossless compression algorithm can be used for prediction and vice versa (Feder & Merhav, 1994). The Prediction by Partial Match (PPM) algorithm, originally developed by Cleary and Witten (1984) and its variant (PPM-C) developed by Moffat, A. (1990)

is considered one of the best lossless compression algorithms. The idea of PPM is to use the last few characters of the input (finite context of K^{th} order, where k represents the number of preceding symbols) to predict the upcoming one. For each K value (K = N to K = -1), K order probabilities of the occurrence of each symbol are computed. During prediction the algorithm starts with the highest order K order model probabilities and if the current input symbol cannot be predicted based on the K^{th} context than the escape probability associated with the K^{th} context is used and the next input symbol is compared against a lower K context until a match is found or eventually stopping at the K-1 context. The probability of an escape event in the PPM-C variant of the algorithm is the proportion of symbols learned for the given K context. The escape represents the penalty incurred for missed predictions. With this approach it was shown that continuing to increase the context length can lead to more accurate predictions but also decreases coverage since there is a greater chance of not giving rise to any prediction at all given that context lengths are associated with many lower valued escape probabilities.

The desire to model large memories uniformly has motivated the need for variable memory models where each element of a sequence is assumed to have a variable number of elements on which it depends. Ron, Singer, and Tishby (1996) have proposed a compact, tree shaped variant of a probabilistic automata called Probabilistic Suffix Trees which allows storage of all substrings of a given string in linear space. A suffix tree representing a sequence *s*, is a rooted tree where each internal node, other than the root has at least two children and each edge is labeled with non empty substring of *s*. A key node property is that no two edges out of a node can have edge-labels beginning with the same character. A key feature of the suffix tree is that for any leaf *i,* the concatenation of the edge labels on the path from the root to the leaf *i* exactly

spells out the suffix of S that starts at position *i.* The algorithm for building a suffix tree is simple: As long as there exists more suffixes of a sequence *s,* add the next shortest suffix to the tree. The authors tested their algorithm by cleaning corrupt text from the bible and showed that suffix trees can capture variable context length predictions by using a compact yet accurate model.

The generative Markov models described so far assume that the probability distribution of an element *i* in the sequence depend only on symbols preceding this element. When the probability distribution of elements in a sequence depends on other factors than a different model must be used. A Markov model where states do not correspond to observed sequence elements but are "hidden" addresses this shortcoming. Hidden Markov Models (HMM) define an emission probability per state that represents the probability of seeing a given element at that state (the emission probability must equal 1 at each state). Computing the probability of generating a sequence is more difficult with HMMs than with Markov chains since with Markov chains a sequence can only be generated through a single path through the states of the model, in contrast, in HMMs a sequence could be generated from an exponential number of paths. This problem makes direct computation of the maximum likelihood of generating a training sequence not feasible, instead HMMs use expectation maximization algorithms like Baum-Welch to estimate the maximum likelihood values needed to generate training sequences.

Validating the Instrument

The prototype that supports the HTM approach proposed in this paper was implemented in Java. This prototype implements the functionality outlined in this chapter. The following self-verification functionality was included in the prototype in order to ensure the reliability of this instrument for the experiments that were conducted for this study:

- Verify the integrity of the connectivity and frequency counts of the Markov graph for each layer of the HTM.
- Verify the integrity of the Markov Chains by verifying that no nodes are shared among any two Markov chains for each layer of the HTM.
- Verify that no ambiguous transitions exist in the Markov graph for each layer of the HTM
- Verify that the clone condition is met for the Markov graphs for each layer of the HTM
- Verify that the sequences learned within the spatial pooler match the sequences represented by the Markov chains since variable order Markov chains can miss-represent learned sequences even when using "cloning" techniques

Validating the Approach

The goal of this study is to recognize a user in the network solely based on prior observed communication behavior independently of the IP address assigned to the source or the complex networks that are traversed by the communication traffic. The following parameters were used to measure this goal: Accuracy and Scalability.

Accuracy was evaluated by:

- Measuring how well the prototype is able to correlate input in the form of individual web destinations to the appropriate session that the input belongs to
- Measuring how well the prototype is able to match sequences of web destinations visited by a specific user

- Measuring how well the Markov chains in the prototype are able to faithfully represent input received, no more, no less.

Accuracy scalability was evaluated by:

- Measuring accuracy as the number of users added to the experiment increases
- Measuring accuracy as the number of destinations visited by users increases

Experiments were conducted using first synthetic data representative of real network data in order to create a baseline of performance, and then real network data was used to further validate the ability of the prototype to satisfy the goals of this study. Experiments that measure accuracy scalability results can be found in section “Accuracy Scalability”.

The baseline was created by evaluating two key tasks that are critical for this study: session identification and user communication behavior attribution. The use of the TCP timestamp option, as proposed in this paper, was used for session identification as a basis of comparison against two other techniques:

1. Time window that predefines a default session duration for each user as proposed in Herrmann et el. (2010). All communication that starts and continues within a time window of time belongs to a given user.
2. Source IP address tracking as proposed in Kumpošt and Matyáš (2009). Source identified by the source IP address assigned to this user each time he/she attaches to a network or each time the address is recycled by the network

The ability to attribute sequences of destinations visited by different users to a given user was accomplished by comparing the HTM approach to traditional generative classification

approaches since these are well understood in the literature and thus represent a good baseline. The following generative classification approaches were considered as a comparison baseline for accuracy against the HTM:

- First Order Markov Model
- Third Order Markov Model
- Finite Context Higher Order Markov Models
    - All-K^{th} Order Markov Model (with K=3) as presented in Pitkow and Pirolli (1999)
    - Prediction by partial match (PPM), using method C algorithm as presented in Moffat, A. (1990) where K = 3

The Algorithm for the First Order Markov Graph that was used for comparison purposes against the HTM is shown in Figure 22.

- Given input stream ***I***, a sequence of web destinations represented as integers, visited by $user_x$
- Build a Matrix "M", initialized with all zeros, where the size of each side is $|\Sigma_{ws}|$ (Σ_{ws} = domain of all web sites visited by any user) that represents the 1st order Markov Graph for $user_x$ such that M[row, column] represents the transition from web site V_{row} to web site V_{column}; i.e. $V_{row} \rightarrow V_{column}$
- During training read the next 2 web sites (V_{row}, V_{column}) from the input stream ***I*** and update the Markov chain probability $P(V_{column} \mid V_{row}) = V_{row}\,V_{column} / V_{row}$ for Markov chain entry M[V_{row}, V_{column}]
- During inference compute the path probability of the input that makes up this observation as follows:
  - Path_probability = 1.0
  - Read next web sites V_K, V_J from input stream
  - While there is more input for this observation Do
    - If M[V_K, V_J] == 0 Then
      - Path_probability = Path_probability * PENALTY
    - Else
      - Path_probability = Path_probability * (M[V_K, V_J] / *SumColumns*(V_K))
    - EndIF
    - $V_K = V_J$
    - Read the next input web site V_J
- Where *SumColumns*(V_K) adds up all the rows at column V_K and PENALTY = $1/ (|\Sigma_{ws}| * |\Sigma_{ws}|)$

Figure 22 First Order Markov Graph Algorithm

Table 10 shows an example of a set of web sessions and the representation of the transition matrix for a third order Markov graph. As in the first order Markov graph the rows of matrix "M" represent the K-order context of the graph. The "M" matrix in Table 10 represents transitions of the form $V_{K1-3} \rightarrow V_{J4}$. The Algorithm for the third order Markov graph that was used for comparison purposes against the HTM is shown in Figure 23. This algorithm computes the maximum likelihood value of parameter $P(V_i \mid V_{i-k} \ldots V_{i-1})$ with K=3 as the ratio of $V_{i-k} \ldots V_{i-1}\,V_i$ occurrences in matrix "M" over the number of $V_{i-k} \ldots V_{i-1}$ occurrences which are computed by summing the frequencies in the $V_{i-k} \ldots V_{i-1}$ row. Note that the sum of the in degree frequencies of node/set of nodes is the same as the sum of the out degree frequencies of that same node/set of nodes except when that node is the last on in the input, in which case the out degree frequency will be one greater than the in degree frequency for that node.

Web Sessions:
{S_1,S_2,S_3,S_4,
S_2,S_3,S_1,S_4,
S_4,S_3,S_1,S_2,
S_1,S_2,S_3,S_4,
S_1,S_4,S_3,S_2,
S_3,S_2,S_4,S_1,
S_1,S_3,S_2,S_4}

$M[V_{K1-3}, V_{J4}]$	S1	S2	S3	S4
$S_1S_2S_3$	0	0	0	2
$S_2S_3S_4$	1	1	0	0
$S_3S_4S_2$	0	0	1	0
$S_4S_2S_3$	1	0	0	0
$S_2S_3S_1$	0	0	0	1
$S_3S_1S_4$	0	0	0	1
$S_1S_4S_4$	0	0	1	0
$S_4S_4S_3$	1	0	0	0
$S_4S_3S_1$	0	1	0	0
$S_3S_1S_2$	1	0	0	0
$S_1S_2S_1$	0	1	0	0
$S_2S_1S_2$	0	0	1	0
$S_4S_1S_4$	0	0	1	0
$S_1S_4S_3$	0	1	0	0
$S_4S_3S_2$	0	0	1	0
$S_3S_2S_3$	0	1	0	0
$S_2S_3S_2$	0	0	0	1
$S_3S_2S_4$	1	0	0	0
$S_2S_4S_1$	1	0	0	0
$S_4S_1S_1$	0	0	1	0
$S_1S_1S_3$	0	1	0	0
$S_1S_3S_2$	0	0	0	1

Table 10 Example of Transition Matrix M for a Third Order Markov Graph

Figure 24 shows the algorithm for the All-Kth Order Markov Graph algorithm with K=3, which combines different context lengths (1-3) in order to improve accuracy. Figure 27 shows the implementation of the Prediction by Partial Match (PPM) algorithm which leverages method "C". The first and third order Markov models were chosen because they represent well understood generative classification algorithms for sequence mining and thus provide a good baseline for comparison purposes with the HTM approach. On the other hand, both the All Kth

Order Markov algorithm and the PPM algorithm were chosen because as generative classification algorithms they provide explicit logic on how to compare input and learned sequences but also detail what to do in cases of a mismatch. In case of mismatch, both of these methods seek shorter prefixes (substrings) of the learned sequences to match the input, whereas the HTM seeks to find the longest common subsequence of learned sequences to match the input. The longest common subsequence is a more forgiving measure of similarity than using a substring. It was important to measure the performance accuracy of these algorithms against the HTM since these algorithms have been successfully used in fields such as data compression and web mining.

- Given input stream ***I***, a sequence of web destinations represented as integers, visited by $user_x$
- Build a Matrix "M", initialized with all zeros, where the columns side is $|\Sigma_{ws}|$ and the row side $|\Sigma_{3ws}|$ (Σ_{ws} domain of all web sites visited by any user and Σ_{3ws} domain of all 3 consecutive web sites visited by $user_x$) that represents the 3rd order Markov Graph for $user_x$ such that M[row, column] represents the transition from web sites $V_{row\ 1\text{-}3}$ to web site $V_{column4}$; i.e. $V_{row1\text{-}3} \rightarrow V_{column4}$
- During training read the next 4 web sites ($V_{row1\text{-}3}$, $V_{column4}$) from the input stream ***I*** and update the Markov frequency counts M[$V_{row1\text{-}3}$, $V_{column4}$]
- During inference compute the path probability ($P(V_{column4} \mid V_{row1\text{-}3}) = V_{row\ 1\text{-}3}V_{column\ 4} / V_{row1\text{-}3}$) of the input that makes up this observation as follows:
  - Path_probability = 1.0
  - Read next 4 web sites $V_{K1\text{-}3}$ (V_{K1}, V_{K2}, V_{K3}), V_{J4} from input stream
  - While there is more input for this observation Do
    - If M[$V_{K1\text{-}3}$, V_{J4}] == 0 Then
      - Path_probability = Path_probability * PENALTY
    - Else
      - Path_probability = Path_probability * (M[$V_{Kk1\text{-}3}$, V_{J4}] / *SumRow*($V_{K1\text{-}3}$))
    - EndIF
    - $V_{K\ 1\text{-}3} = V_{k2\text{-}3}\ V_{J4}$
    - Read the next input web site V_{J4}
- Where *SumRow*($V_{K1\text{-}3}$) adds up row $V_{K1\text{-}3}$ of matrix **M** (see example in Table 10) and PENALTY = 1/ ($|\Sigma_{ws}| * |\Sigma_{3ws}|$)

Figure 23 Third Order Markov Graph Algorithm

- Build a 1st order Markov graph M_1 as shown in Figure 22
- Build a 2nd and 3rd order Markov graph M_2, M_3 as shown in Figure 23
- During training read input and update frequencies in all three Markov graphs (M_1, M_2, M_3)
- During inference try and match input against the following states of each Markov graph from the highest order (M_3) to the lowest order (M_1) until a match is found for:
  1. M_3: $V_{K1\text{-}3}$ (state representing the third order context for this Markov graph)
  2. M_2: $V_{K1\text{-}2}$ (state representing the second order context for this Markov graph)
  3. M_1: V_K (state representing the first order context for this Markov graph)
- As soon as a match is found for a given state in the Markov graph, compute the path probability of the input sequence just read based on the logic specific to the order of the matched graph using Figure 23 if the input matched a state in M_2 or M_3, otherwise using Figure 22 if the input matched a state in M_1

Figure 24 All-Kth Order Markov Algorithm, where K = 3

In addition, algorithms operating at a layer 1 of the HTM were compared to algorithms operating at layers 2 and 3 of the HTM in order to explore the merits of the hierarchical structure of HTM. Accuracy and scalability were assessed for all experiments conducted against the baseline.

Scalability in this study represents the ability of the solution to provide consistent levels of accuracy during training and inference as the number of users increases while keeping constant the number of destinations and vice versa increasing the number of destinations as the number of users remains constant. The ability to be accurate and scalable in the solution to the user attribution problem has been achieved in a very limited fashion by the work of Herrmann, et al. (2010) who used a set of 28 users from a real world data set. These authors explain that the rigid time window used to identify sessions belonging to the same user prevents their solution from scaling to a higher number of users since their approach will erroneously distribute contiguous requests across two sessions when these sessions cross time boundaries. Yang (2010), leveraged a maximum of 100 users in her experiments and also acknowledges the limitations of being able to identify consecutive user sessions being forced to use short periods of user web activity to track sessions. Banse et al. (2012) are able to scale their solution to much higher numbers, 2100 concurrent users on average, while still using a fixed time window. Their solution scales better due to the tracking strength of source IP addresses, not assumed to change within a fixed time window, assigned to a given user. The authors acknowledge the scalability limitations of their solution when source IP addresses change often as is the case when a user moves across mobile networks.

In this study, scalability was assessed by increasing the number of users from 5, 20, 50, 100, up to 500 and by increasing the number of web sites from 1000, 5000, up to 10,000 (for experiment results which measure scalability based on these parameters see the "Results" chapter with specific emphasis on section "Results of experiments to verify user attribution accuracy without concept drift using synthetic data") . The range of the number of users and web sites utilized was enough to show how accuracy improves, remains constant or deteriorates when both the number of users and/or web sites increases. Experiments were also conducted with 150, 250, 350 and 450 users as reported in section "Accuracy Scalability", utilizing the best performing HTM algorithms at layers 1 and 3, in order to provide even more insight into accuracy scalability as it specifically pertains to increasing number of users when the number of destinations remains fixed.

The limit of 500 as the maximum number of users is the result of the challenge that these experiments encounter in the computer run time and memory needed to complete the training and inference stages for all experiments since there are seven algorithms supported by the HTM, plus four alternate approaches, all of which need to be verified against the different combination of number of users and web sites described above using appropriately sized training and test data sets.

-During Training for each context of size K=3 down to K=1 Do

- J = K
- Get |K| + 1 input symbols: $\mathbf{i_{j-2}i_{j-1}i_j\ i_{j+1}}$, where $1 \le j \le N$, N = number of symbols in input
- Record K input symbols $\mathbf{i_{j-2}i_{j-1}i_j}$ into context $\mathbf{C_{k,j}} = \{ \mathbf{i_{j-2}i_{j-1}i_j} \}$
- While there is more input to be read Do
  - Record input symbol $\mathbf{i_{j+1}}$ following the |K| symbols as transaction $\mathbf{C_{k,j} \rightarrow i_{j+1}}$
  - Update the frequency count $\mathbf{F_{Ckj,i_{j+1}}}$ for transaction $\mathbf{C_{k,j} \rightarrow i_{j+1}}$
  - Update context $\mathbf{C_{k,j}}$ by sliding the context to the right of the input by one symbol $\mathbf{C_{k,j}} = \{ \mathbf{i_{j-1}i_j\ i_{j+1}} \}$
  - j = j + 1
  - Get the next input symbol $\mathbf{i_{j+1}}$

EnDo

For each recorded transaction $\mathbf{C_{k,j} \rightarrow i_{j+1}}$ Do

-Add an escape symbol € to the set of symbols following context $\mathbf{C_k}$ with a frequency count $\mathbf{F_{Ck,j€}}$ = M, where **M** = Number of symbols that follow context $\mathbf{C_{k,j}}$

-Compute $\mathbf{P(i_{j+1} \mid C_{k,j}) = F_{Ckj,i_{j+1}} / (\sum_1^M F_{Ckj,i_{j+1}}) + F_{Ck,j€}}$

-Compute $\mathbf{P(C_{k,j€}) = F_{Ck,j€} / (\sum_1^M F_{Ckj,i_{j+1}}) + F_{Ck,j€}}$

EnDo

EnDo

During training when K=0, $\mathbf{C_k} = \{\}$ and $\mathbf{C_k \rightarrow i_\Sigma}$, where each $\mathbf{i_\Sigma}$ is a unique input symbol in the input Σ Alphabet

-Add an escape symbol € to the set of symbols following context $\mathbf{C_k}$ with a frequency count $\mathbf{F_{Ck€}}$ = M

-Compute $\mathbf{F_{Ck,i_\Sigma}}$ as the number of times $\mathbf{i_\Sigma}$ occurs in the input

-Compute $\mathbf{P(i_\Sigma) = F_{Ck,i_\Sigma} / (\sum_1^M F_{Ck,i_\Sigma}) + F_{Ck€}}$

-Compute $\mathbf{P(C_{k€}) = F_{Ck€} / (\sum_1^M F_{Ck,i_\Sigma}) + F_{Ck€}}$

EnDo

Figure 26 Training Algorithm for Prediction By Partial Match (PPM) using Method C

- Initialization: Match not Found; K = 3; i = 1; J = K; path_probability = 1.0;
- Get |K| + 1 input symbols: $\mathbf{i_{i..j}}$, $\mathbf{i_{j+1}}$
- While there is more input to be read Do
  - While Match not Found AND k >= 0 Do
    - IF i ≤ j AND input symbols $\mathbf{i_{i..j}}$ match any context symbols in $\mathbf{C_k}$
      AND $\mathbf{i_{j+1}}$ matches symbol following $\mathbf{C_{k,j}}$ ($\mathbf{C_{k,j\rightarrow}}$ $\mathbf{i_{j+1}}$)
    - THEN // Compute the probability of the symbol following matched context
      - // When K = 0, $P(i_{j+1} \mid C_{k,j}) = P(i_{j+1})$
      - path_probability = path_probability * $\mathbf{P(i_{j+1} \mid C_{k,j})}$
      - Match Found
    - ELSE IF (i ≤ j AND input symbols $\mathbf{i_{j..i}}$ match any context symbols in $\mathbf{C_k}$
      - AND $\mathbf{i_{j+1}}$ Does NOT match symbol following $\mathbf{C_{k,j}}$ ($\mathbf{C_{k,j\rightarrow}}$ $\mathbf{i_{j+1}}$))
      - OR (i > j AND $\mathbf{i_{j+1}}$ Does NOT match symbol following $\mathbf{C_{k,j}}$ ($\mathbf{C_{k,j\rightarrow}}$ $\mathbf{i_{j+1}}$))
    - THEN
    - // No match for symbol following context ➔ use escape probability
    - // and match shorter prefix of the context (lower k order context)
      - path_probability = path_probability * $\mathbf{P(C_{k,j\epsilon})}$
      - k = k – 1
      - i = i + 1
    - ELSE // Input does not match context ➔ match a lower K order context
      - k = k – 1
      - i = i + 1
    - ENDIF
  - EnDo
  - IF Match not Found
    - THEN
      - // K = -1 , Input did not match any prefix of context or learned single symbol
      - path_probability = path_probability * **(1/|Σ|)**
    - ENDIF
    - // Drop leftmost input symbol from input, keeping input size = k and read next symbol
  - i = i + 1
  - j = j + 1
  - Read next input symbol $\mathbf{i_{j+1}}$

EnDo

Figure 27 Inference Algorithm for Prediction By Partial Match (PPM) using Method C

It is important to note that a prediction by partial match (PPM) can also be implemented by using an All-K algorithm (the implementation chosen for this study) and representing escape frequencies for each k order context (K > 0) as an additional column in each K Markov graph as shown in Table 11.

Web Sessions:
{S_1,S_2,S_3,S_4,
S_2,S_3,S_1,S_4,
S_4,S_3,S_1,S_2,
S_1,S_2,S_3,S_4,
S_1,S_4,S_3,S_2,
S_3,S_2,S_4,S_1,
S_1,S_3,S_2,S_4}

K=3 Context	S1	S2	S3	S4	Esc
$S_1S_2S_3$	0	0	0	2	1
$S_2S_3S_4$	1	1	0	0	2
$S_3S_4S_2$	0	0	1	0	1
$S_4S_2S_3$	1	0	0	0	1
$S_2S_3S_1$	0	0	0	1	1
$S_3S_1S_4$	0	0	0	1	1
$S_1S_4S_4$	0	0	1	0	1
$S_4S_4S_3$	1	0	0	0	1
$S_4S_3S_1$	0	1	0	0	1
$S_3S_1S_2$	1	0	0	0	1
$S_1S_2S_1$	0	1	0	0	1
$S_2S_1S_2$	0	0	1	0	1
$S_4S_1S_4$	0	0	1	0	1
$S_1S_4S_3$	0	1	0	0	1
$S_4S_3S_2$	0	0	1	0	1
$S_3S_2S_3$	0	1	0	0	1
$S_2S_3S_2$	0	0	0	1	1
$S_3S_2S_4$	1	0	0	0	1
$S_2S_4S_1$	1	0	0	0	1
$S_4S_1S_1$	0	0	1	0	1
$S_1S_1S_3$	0	1	0	0	1
$S_1S_3S_2$	0	0	0	1	1

Table 11 All-K Implementation of PPM

Each escape frequency equals the number of non-zero columns for a given k context. The escape frequency for a K=0 context equals the number of contexts in a given Markov graph (21 for the Markov graph in Table 11). The *SumRow* in Figure 23 would also need to be adjusted to

account for escape frequencies. Finally, the All-K algorithm which normally represents K > 0 order Markov graphs would now need to also support a K=0 Markov graph.

Generation of Synthetic Data for the Simulation

Training and test data was produced based on two different HTM verification approaches:

- The HTM was verified against synthetic data that mimics user web visits found in real world scenarios as shown in Figure 28, using the algorithm presented in Figure 29 and Figure 30
- The HTM was verified against other approaches as presented in the "Validating the Approach" section. These approaches, as opposed to the HTM, do not leverage any timing information. For these tests the same synthetic data generated by the algorithm in Figure 29 was used to generate input for all these alternate approaches but 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. During inference all approaches, including the HTM approach, output the specific inferred user on a per observation basis.

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:

- 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.
- 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).
- Generation of destinations (ranked in order of popularity) visited by all users in the simulation followed a power law distribution (Zipf)

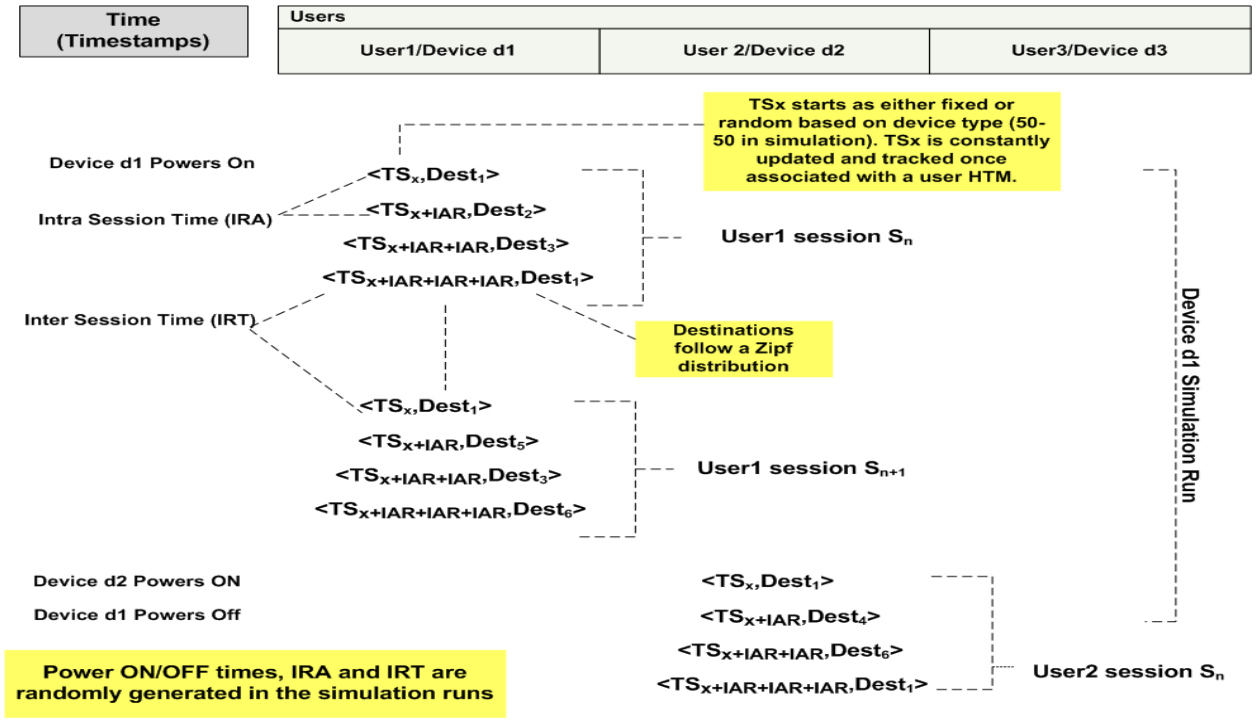


Figure 28 Synthetic Input Data for User Attribution Simulation

Figure 28 shows the input framework within which the simulation was run. A Java application was developed separate from the prototype, which produced, for each user, the synthetic input data as shown in Figure 28. The data simulated devices associated with users entering the network at random times and initiating multiple communication sessions until the

devices are turned off. The table 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 12 Simulation Parameters

The input generation application creates an input file for each simulated user where the number of simulation runs is a configurable parameter of the application and determines the number of power on/off cycles that each device is allowed for a given simulation.

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

// 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
        While (Ux_Max_Simulation_Days   > 0 ) Do
            DevicePowerOnTime  =        TimeStamp              + Uniform Random(0, 3Hrs)
            DevicePowerOffTime =        DevicePowerOnTime      + Uniform Random(0, 21Hrs)
            TS                 =         Generate TCP TimeStamp-TS
            TimeStamp          =        DevicePowerOnTime


            While (TimeStamp  <  DevicePowerOffTime) Do
                    NumberDestinationsPerSessions =  Uniform Random(1,10)
                    While (NumberDestinationsPerSessions  > 0 AND TimeStamp  <  DevicePowerOffTime) Do

                            Dest    =  Next_ZipfRandom (1000,theta)
                            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(1,40mins)
                    TimeStamp             = InterSessionTime-IRT
                    TS                    = TS  + InterSessionTime
            EnDO

            Ux_Max_Simulation_Runs     = Ux_Max_Simulation_Runs – 1
        EnDO
```

Figure 29 Algorithm to generate synthetic random train input for a single user

The above algorithm creates 5 simulation days' worth of synthetic data for user Ux. The simulation code in Figure 29 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, a power law based distribution, with the most popular destinations having the highest probability of being selected over less popular destinations. Only web destinations are recorded for alternate approaches since they do not rely on time.

The synthetic data created by the input generator for each user was merged in order to simulate a real world scenario where many users enter and exit the network concurrently as shown below.

```
// Merge user files in timestamp order in to a single file which includes input from all users
// Each user input user file produced by algorithm in Figure 29 is stored in a fnameUx
For Each user file fnameUx in simulation Do
        Read and Save the next time stamped input Timestamp<TS, Dest> from file fnameUx
EnDO

Sort saved time stamped inputs in ascending order of TimeStamp
Output sorted time stamped input Timestamp<TS, Dest> and append to input  filename  fnameUx
```

Figure 30 Algorithm to generate synthetic random input for multiple users

Why do we need to append the file name **(**Timestamp<TS, Dest> **fnameU$_x$)** to the input?

The fnameU$_x$ is completely ignored by the prototype during all phases of learning and inference. The fnameU$_x$ is used only to validate the accuracy of the HTM$_{UX}$ in recognizing users. Each time the HTM is fed an input token such as Timestamp<TS, Dest>fnameU$_x$, the HTM saves the received input in an HTM$_{UX}$ specific output file. After the simulation is run, a scan of the HTM$_{UX}$ specific output file allows determination of false positives (mistaken users) since all

input tokens in the HTM_{UX} output file produced by the HTM for user x should contain Timestamp<TS, Dest>$fnameU_x$ where $fnameU_x$ is user x. On the other hand, running the *diff* utility between $fnameU_x$ and the HTM_{UX} files allows identification of false negatives (users that were missed; i.e. not recognized by the HTM_{UX}).

Test Data needs to be created using a different approach since it must be similar to the train data but also maintain a certain level of independence from train data. Three methods are used for generation of synthetic data for the test phase of experiments. All three algorithms walk a first order Markov chain of learned destinations which were generated by the input generator based on the algorithm in Figure 29.

- **Random Walk** –The next destination $\boldsymbol{V_j}$, for transitions of the form $\boldsymbol{V_i \rightarrow V_j}$, is chosen randomly in proportion to the in-degree of the node $\boldsymbol{V_j}$. That is, in proportion to the access frequencies of the neighbors ($\boldsymbol{V_{j1}, \ldots V_{jn}}$) of the current node ($\boldsymbol{V_i}$). If no such neighbor $\boldsymbol{V_j}$ exists then the walk proceeds with a new node $\boldsymbol{V_i}$ with at least one neighbor, selected from the learned destinations based on a zipf distribution. Selection of the next destination $\boldsymbol{V_j}$ is based on the work of Price (1976) 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 is presented below:

- Follow with connectivity probability **1 -** $\frac{Oi}{Oi+Ci}$ **> r (0 ≤ r ≤ 1)** a learned path proportional to the frequency of the in-degree of web sites along the path. Otherwise start a new path. **r** is a random number that follows a uniform distribution**.**
- $\mathbf{C_i}$ = Sum of traversal frequencies of all edges emanating from V_i $(V_i \rightarrow V_{j1\text{-}n})$
- $\mathbf{O_i}$ = 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".

- **Walk Only** - 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**.
- **Context Drift** – 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).

Figure 31 shows the entire process used to generate synthetic train and test data for simulations. CSV files are coma delimited files that just record web destinations. They are used for two purposes. For alternate algorithms, CSV files represent train and test input files. In addition, CSV files are also used to match the output of the HTM and the output of alternate algorithms against the original test files generated for each experiment.

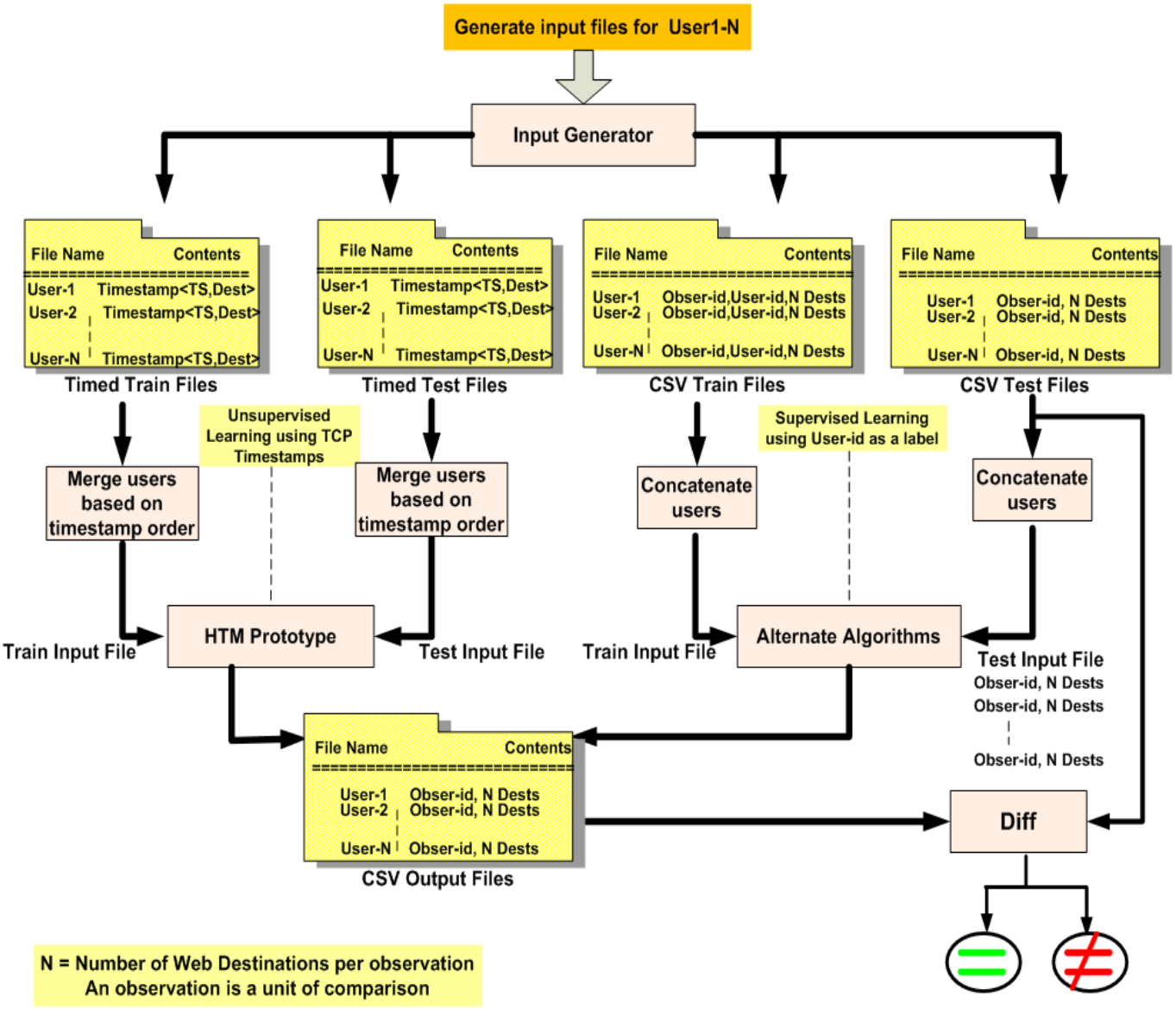


Figure 31 Synthetic Data Generation Process

Generating Malicious Data for the Experiments

Simulation of malicious data was used in this study to measure how well the HTM can recognize malicious users reentering the network. Two types of malicious attacks were simulated for these experiments: Phishing and Denial of Service attacks.

- Phishing attacks were modeled by simulating few malicious phishing web sites dedicated to download of software that performed the phishing attack, as well as, web sites that actually carry out the actual phishing attack (e.g. as for a commercial on-line bank). The simulation data would contain few users that participate in the phishing attack with the rest of the users being non malicious. A small portion of the non malicious users would accidently visit the web sites that actually carry out the actual phishing attack.
- DOS attacks were modeled as a small group of users that visit the same site with high persistence within a short period of time.

The attack data produced for these experiments by a given malicious user or compromised device was embedded within normal usage data for these users.

Resource Requirements

All experiments that use synthetic data utilized a standard laptop computer for building the prototype and for building the algorithms to produce the synthetic data. All experiments that utilize real network data required access to operator network subscriber data via a carrier grade network traffic collector. The operator used for these experiments is U.S. Cellular and with their permission the Wireless Network Guardian (manufactured by Alcatel Lucent) traffic collector was used to collect live traffic data that was used for the experiments.

Chapter 4

Results

Results of experiments to verify user attribution accuracy without concept drift using synthetic data

A main objective of this study is to address the user attribution problem as accurately as possible in terms of the ability of the HTM to be able to correctly identify sequences of web destinations visited by users over time. Test results were recorded in terms of recall (number of correctly matched destinations for this user as a fraction of all possible correct observations for this user), precision (number of correctly matched destinations as a fraction of all destinations matched for this user during the experiment) false positives and false negatives. So, if the HTM matched 80 observations for user-x such that 80 observations for user-x are correctly matched out of a total of 80 possibly correct observations for this user then recall and precision equal 100%. However, if the HTM matched 100 observations for user-x such that 80 observations for user-x are correctly matched out of 80 possibly correct observations for this user then recall is 100% but precision is 80%. In this chapter *recall* is used to report accuracy keeping in mind that Appendix A through Appendix H contain the rest of the statistics collected (recall, precision, false positives and false negatives).

Tests were conducted that measured the ability of the HTM and alternative approaches to scale accuracy by maintaining high levels of recall and precision as the number of users and the number of destinations increased.

Thirty sets of experiments each run with 11 different algorithms, 7 HTM algorithms (Simple Average, BottomUp, Path Probability for layers 1 and 3 and TopTop for layer 3) and 4 Alternate algorithms (1st and 3rd order Markov chains, All-K with K=3 and Partial Prefix Match) were conducted as shown in Table 13. Each square in Table 13 represents execution of 11 experiments using synthetic data with parameters based on different combinations of web destinations (1000, 5000, 10000) and users (5, 20, 50, 100, 500). Synthetic data was generated for both train and test data sets based on the algorithms described in "Generation of Synthetic Data for the Simulation". Train data sets were limited to 5 train days' worth of data while the test data set ranged from 1 day worth of test data, to 3 observations (150 web destinations) worth of test data. Squares with a red cross indicate experiments that were not executed. For a full description of the results of these experiments see Appendix A. The rest of the discussion in this section only reports a key subset of the overall results from Appendix A in order to determine how well the goals of this study were met.

Number Destinations	5 Users	20 Users	50 Users	100 Users	500 Users
1000	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	
	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only
5000	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	
	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only
10,000	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	5/1, Walk_Only, No CD	

Number Destinations	5 Users	20 Users	50 Users	100 Users	500 Users
	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only	3 obs, Walk_Only, No CD, HTM Layer1 only

Table 13 Accuracy tests completed using Synthetic data with no concept drift

The experiments results in Figure 32 used synthetic data, (without any concept drift) and simulated user web visits over time periods of 5 train days and 1 test day for 1000 web destinations with a range of users from, 5, 20, 50, and 100. For the purpose of the following discussions only the recall measurement are reported (the rest of the measurements and experiments can be found in Appendix A) to compare accuracy results.

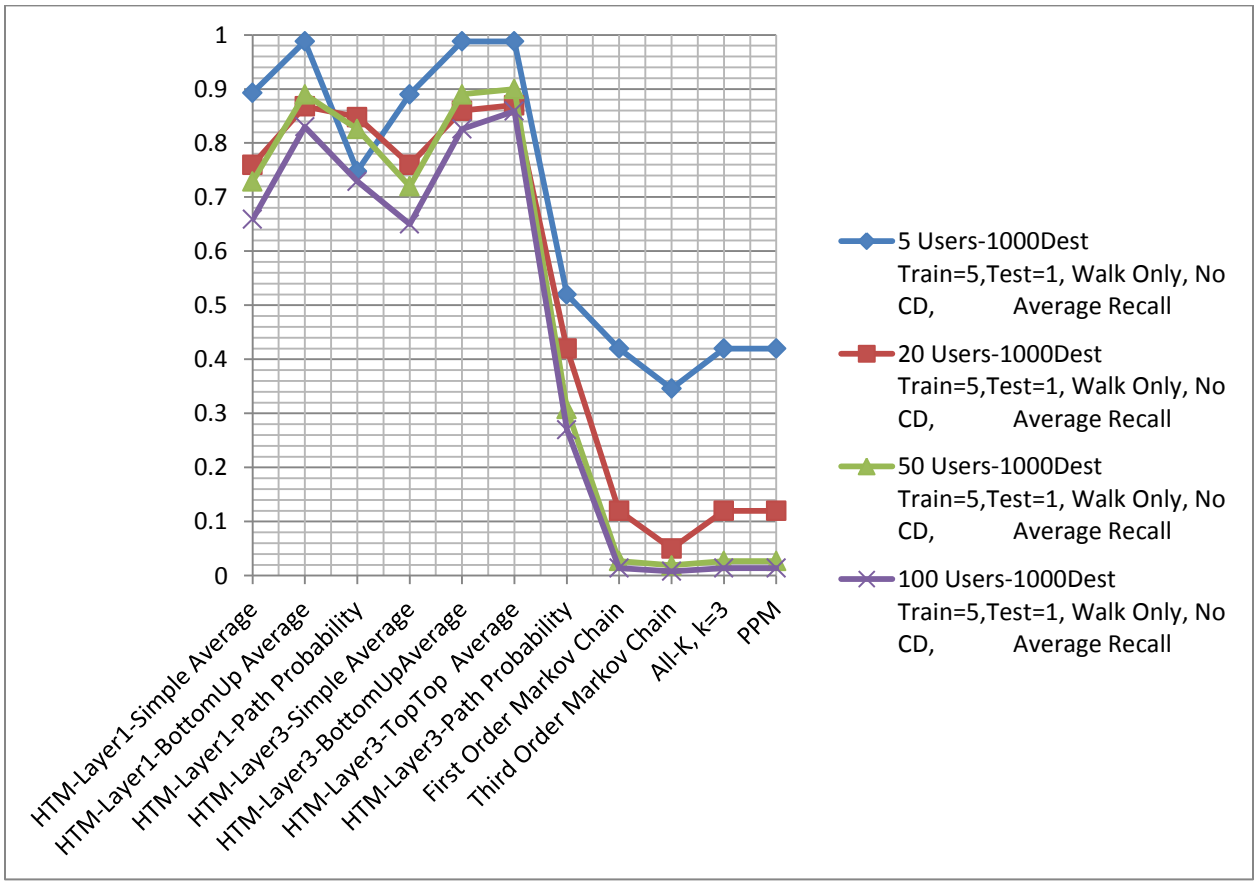


Figure 32 Experiment 5-100 users, 1000 Destinations, 5 Train Days and 1 Test Day

The graph above shows the following:

- Alternate algorithms (1st and 3rd Order Markov Chain, All-K and PPM) perform very poorly in terms of accuracy compared to HTM algorithms
- Accuracy for alternate algorithms scales poorly as the number of users increases in the experiment
- HTM algorithms are considerably more accurate than alternate algorithms
- Recall accuracy for HTM algorithms scales better than for alternate algorithms
- Recall accuracy reported by HTM algorithms at layers 1 and 3 is comparable
- HTM algorithms Bottom Up and TopTop perform the best among all HTM algorithms
- HTM path probability at layer 3 is the least accurate of the HTM algorithms
- Algorithm 3rd Order Markov Chain is the least accurate of the Alternate algorithms

Consider what happens when the number of web destinations visited increases from 1000, as in the previous experiment, to 5000 and then 10,000 respectively as shown below in Figure 33 and Figure 34. The accuracy of all HTM algorithms increases in line with scalability, while no improvement can be seen for the alternate algorithms. Specifically, for 100 users HTM algorithms Bottom up and TopTop perform in the range from 97% to 99% accuracy a big improvement when compared with 86% accuracy reported by the same algorithms for experiments with 1000 destinations.

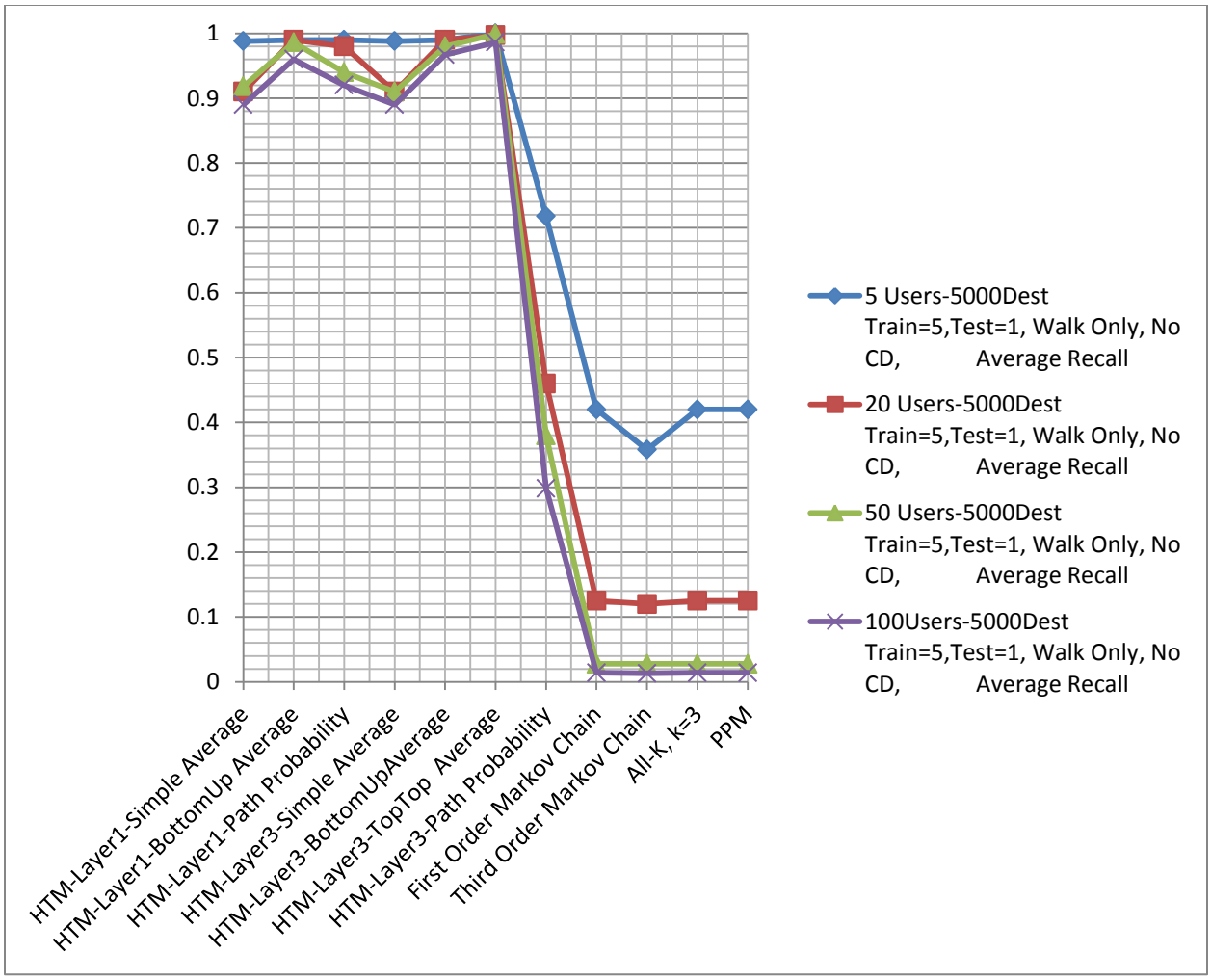


Figure 33 Experiment 5-100 users, 5000 Destinations, 5 Train Days and 1 Test Day

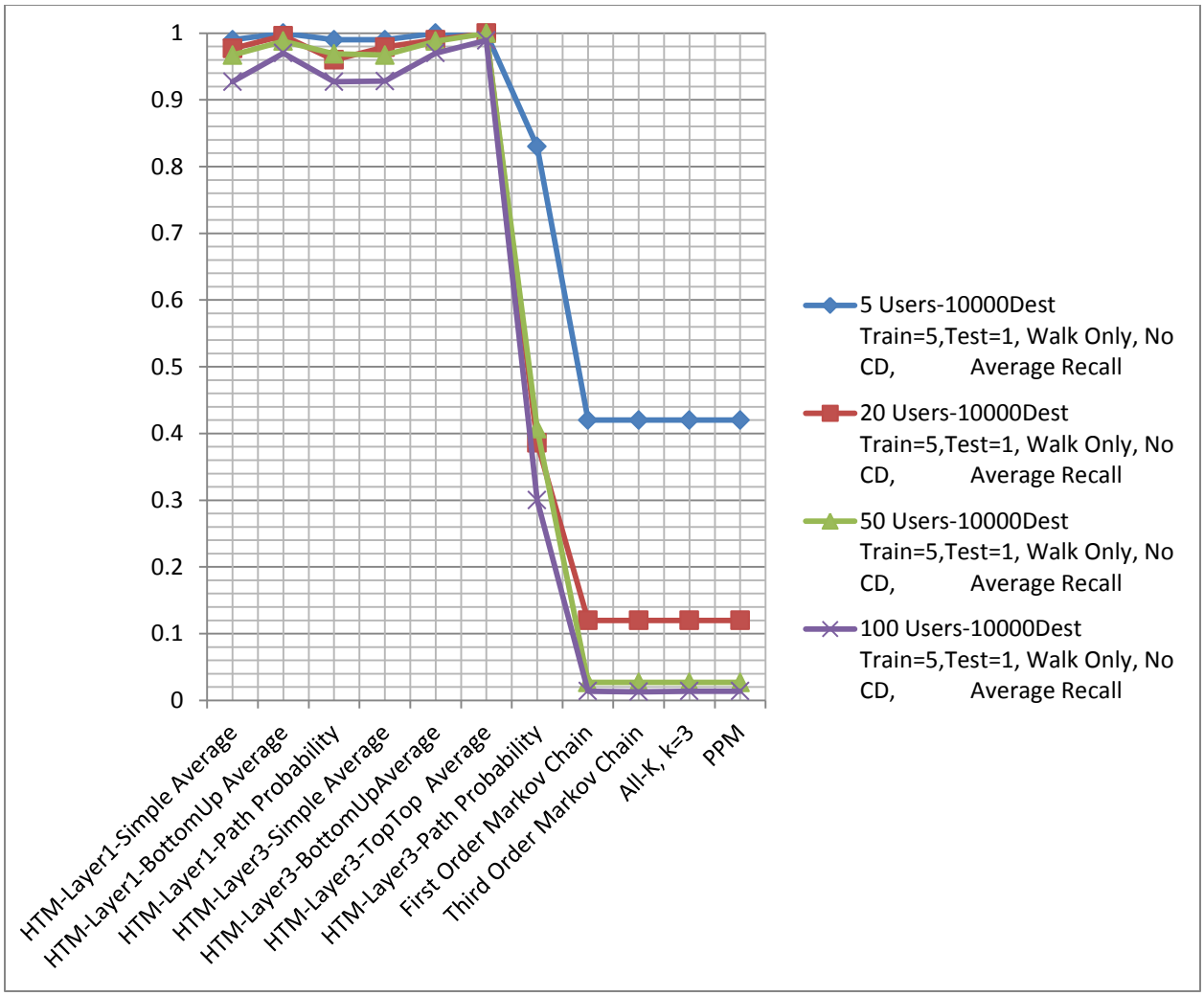


Figure 34 Experiment 5-100 users, 10,000 Destinations, 5 Train Days and 1 Test Day

Why are the accuracy results for alternate algorithms so poor? Is it possible that these algorithms were not correctly implemented? To answer these questions it is important to note that all algorithms used for the experiments were calibrated. That is, each algorithm was trained with a given data set and then it was fed that same data set as test data. The expected behavior is that correctly implemented algorithms can recognize their own learned input. For alternate algorithms Table 14 shows perfect accuracy for all algorithms as a result of calibration using synthetic input data. Calibration on all HTM algorithms for synthetic data with the same input

show similar results (see Appendix H). From these experiments the following can be stated about the calibration process:

1. A well calibrated algorithm works as intended as is capable of recognizing its own trained input from other input
2. A well calibrated algorithm will not necessarily perform well when inferring from test input that differs from the training input
3. In this study, calibration is used to baseline an algorithm as being implemented correctly with respect to its abilities to appropriately train and infer its own input.

Users	**Alternate Approaches % Accuracy**
	1st Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	3rd Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	All K Order Markov (K=3)
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	PPM
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%

Table 14 Alternate Algorithms Calibration results for 5 users, 5 Train Days and 2 Test days

The accuracy reported for HTM algorithms in these experiments is quite high, is it possible that the synthetic train and test data sets created by the input generator are very similar to each other and would thus allow the HTM algorithms to perform at very high levels of accuracy?

All synthetic input generated always reports the following similarity statistics, derived from the work of Kumar, Krishna, and Raju (2010), between train and test data sets generated based on all observations processed.

Sequence Similarity =

$$\sum_{\mathbf{1}}^{\boldsymbol{observations}} \frac{\boldsymbol{LCSL(Tes\ Observation,\ \ Train\ Observation)}}{\boldsymbol{Size\ of\ Test\ observation}} \Big/ \boldsymbol{Num\ Users}$$

Substring Similarity =

$$\sum_{\mathbf{1}}^{\boldsymbol{observations}} = \frac{\boldsymbol{LCSSL(Test\ Observation,\ \ Train\ Observation)}}{\boldsymbol{Size\ of\ Test\ Observation}} \Big/ \boldsymbol{Num\ Users}$$

Set Similarity =

$$\frac{(\boldsymbol{Test\ Observation} \cap \boldsymbol{Train\ Observation})}{\boldsymbol{Size\ of\ Test\ Observation}}$$

Total Similarity = (.33) Sequence Similarity + (.33) Substring Similarity + (.33) Set Similarity

LCSL is the length of the longest common subsequence between train and test observations, whereas LCSSL is the length of the longest common substring between train and test observations. As can be seen from Figure 35 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 Figure 35) for an equivalent data set (5 users, 5 train days, and 1 test days) collected from a real network show that synthetic data similarity measurements are in line with real data but show less similarity than real data between train and test sets.

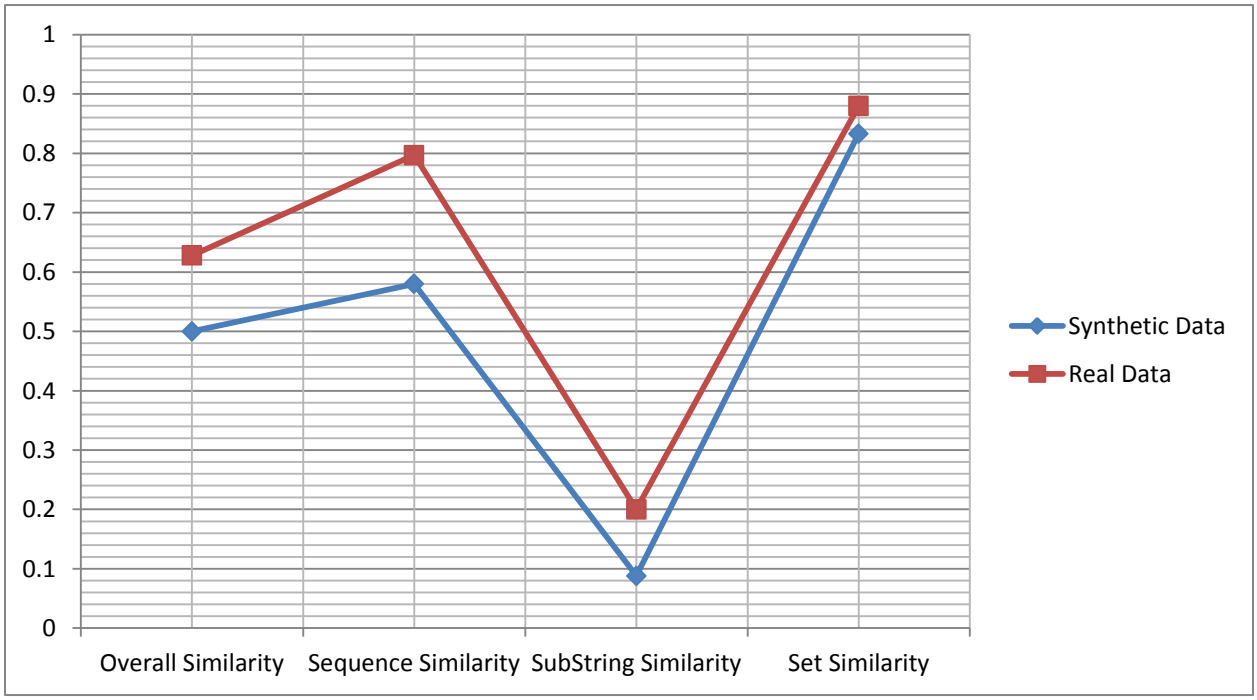


Figure 35 Similarity Stats for synthetic data for 5 users, 5 Train Days and 1 Test Day

These results indicate that synthetic test data is relevant enough to the train data set while maintaining enough independence from the test data set to support realistic experiments.

The next set of experiments uses synthetic data but extends the number of users to 500 and limits the number of observations (each containing 50 destinations) in the test data set to 3. The reason for limiting the test data set to only three observations was due to two key reasons:

1. In security scenarios user attribution needs to be performed as quickly as possible, using as few observations as possible. This is different from the previous experiment where an entire day worth of observations was used for the test data set.
2. The time to complete experiments using the HTM increases dramatically (as high a 5 and a half hours for a single HTM algorithm run) as the number of user reaches 500.

Figure 36 shows the results of running the experiment with 3 test observations with 1000 destinations. While accuracy continues to be better for the HTM versus alternate algorithms, the overall HTM accuracy is poor especially for 500 users. These tests also show that the HTM does not scale well moving from 100 to 500 users.

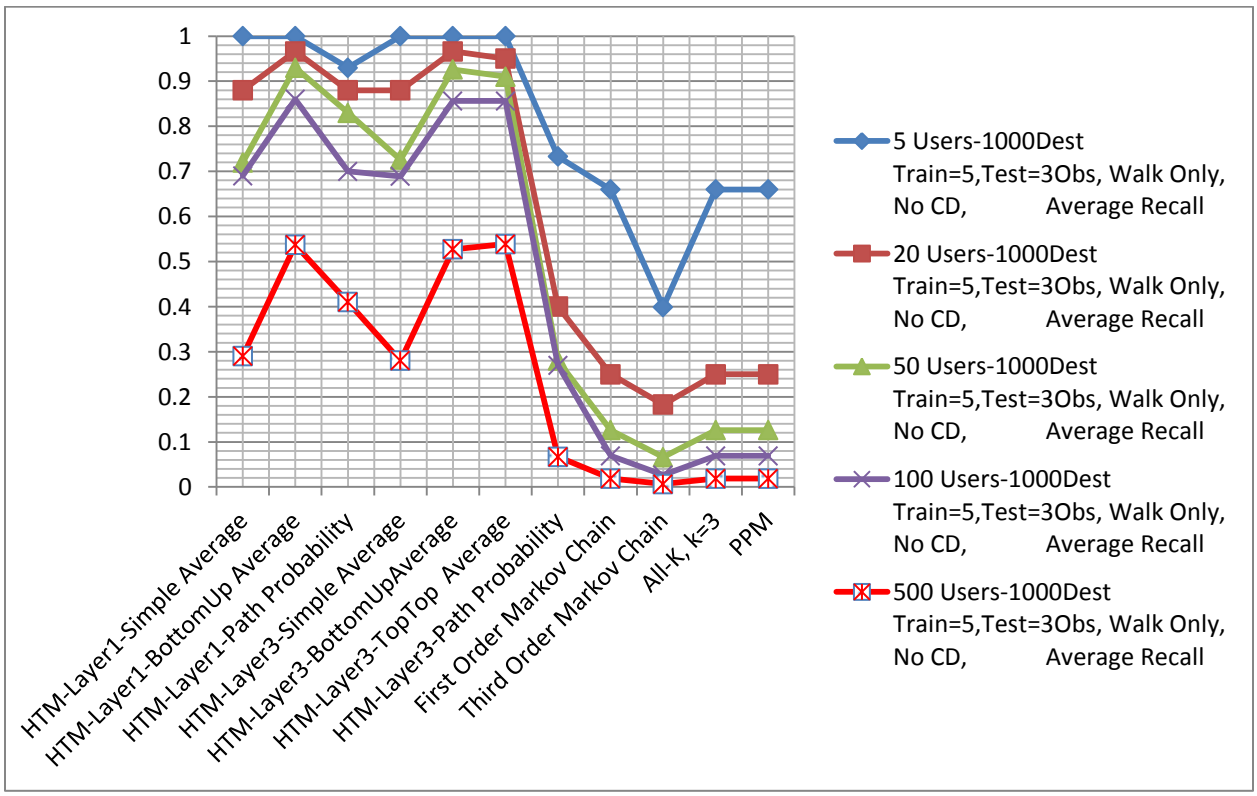


Figure 36 Experiments for 5 Train days, 3 Observations for test data, 1000 destinations

Figure 37 shows the same experiments but this time the number of destinations in the train and test data sets is increased from 1000 to 5000. These results are quite different from the

previous results using 1000 destinations and are more in line with the results obtained for the experiments using a 1 day worth of test data observations with 5000 destinations. HTM algorithm Bottom up and TopTop continue to outperform all other algorithms. HTM algorithms scale well even for 500 users with accuracy as high as 99% for the TopTop algorithm. Alternate algorithms continue to underperform HTM algorithms.

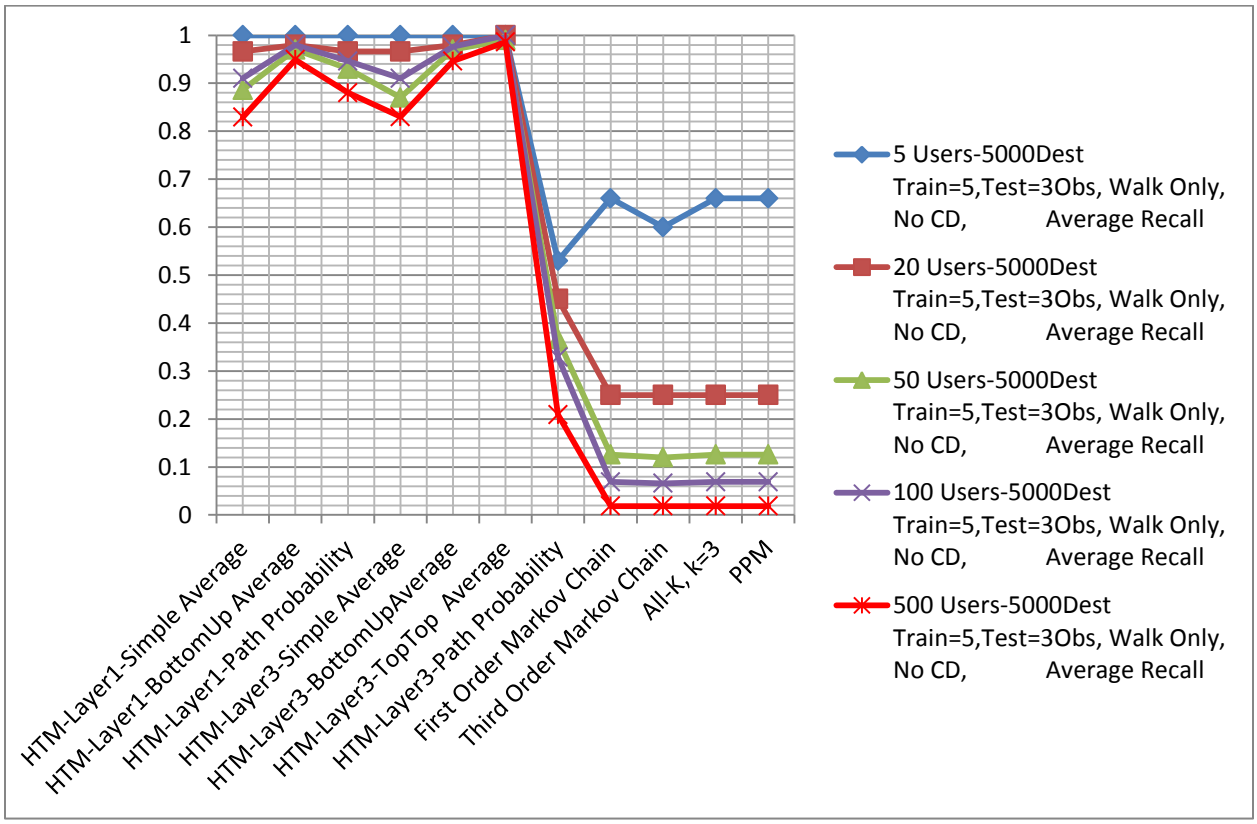


Figure 37 Experiments for 5 Train days, 3 Observations for test data, 5000 destinations

Figure 39 shows the results of further extending the number of web destinations allowed in the train and test data sets to 10000. The same conclusions can be drawn for these results as for the previous ones with experiments conducted using 5000 destinations.

While HTM algorithms are more accurate, they take longer to provide results than the alternate algorithms. Why? The HTM performs an *exhaustive* search of all Markov chains when presented with an input and finds the best matching longest common subsequence and substring that was observed most often during training. In contrast all alternate algorithms are based on extensions of a 1st order Markov graph which matches the input completely based on the on the very first destination in the sequence. Alternate approaches find this first destination in constant time and then match the rest of the input from that point in the graph onward. The accuracy superiority of HTM algorithms is due to the extensive search across all Markov graphs learned during train time at each layer of the HTM which allows HTM algorithms to find the best match for the input in contrast to the alternate algorithms which only search a very small subset of the train data set resulting in a lot of mismatches.

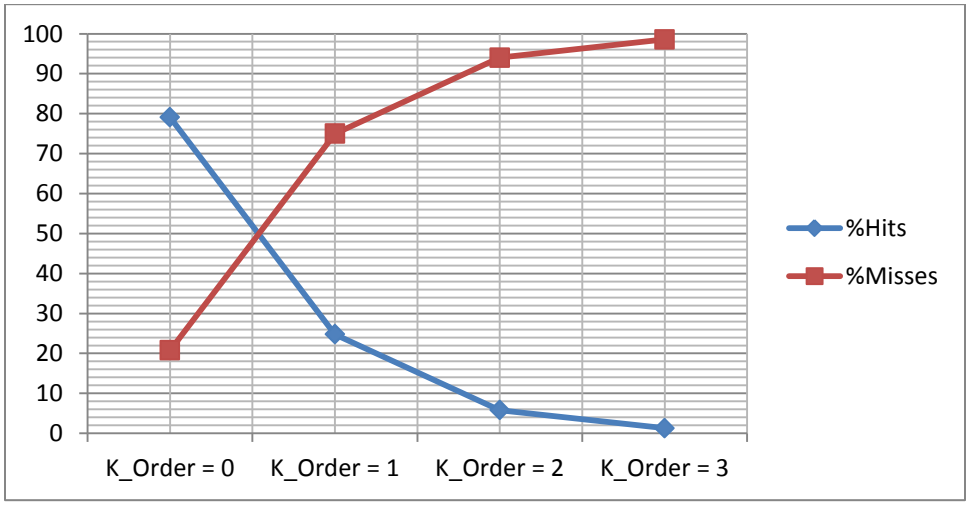


Figure 38 PPM Matches and Miss Matches per K-Order = 3

Figure 38 shows the PPM statistics for the experiment run in Figure 37. 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). Figure 38 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).

Alternate 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 occurs in time proportional to the size of the input. HTM algorithms on the other hand have search run times that are proportional to the size of the entire input learned at training time as well as the size of the input sequence.

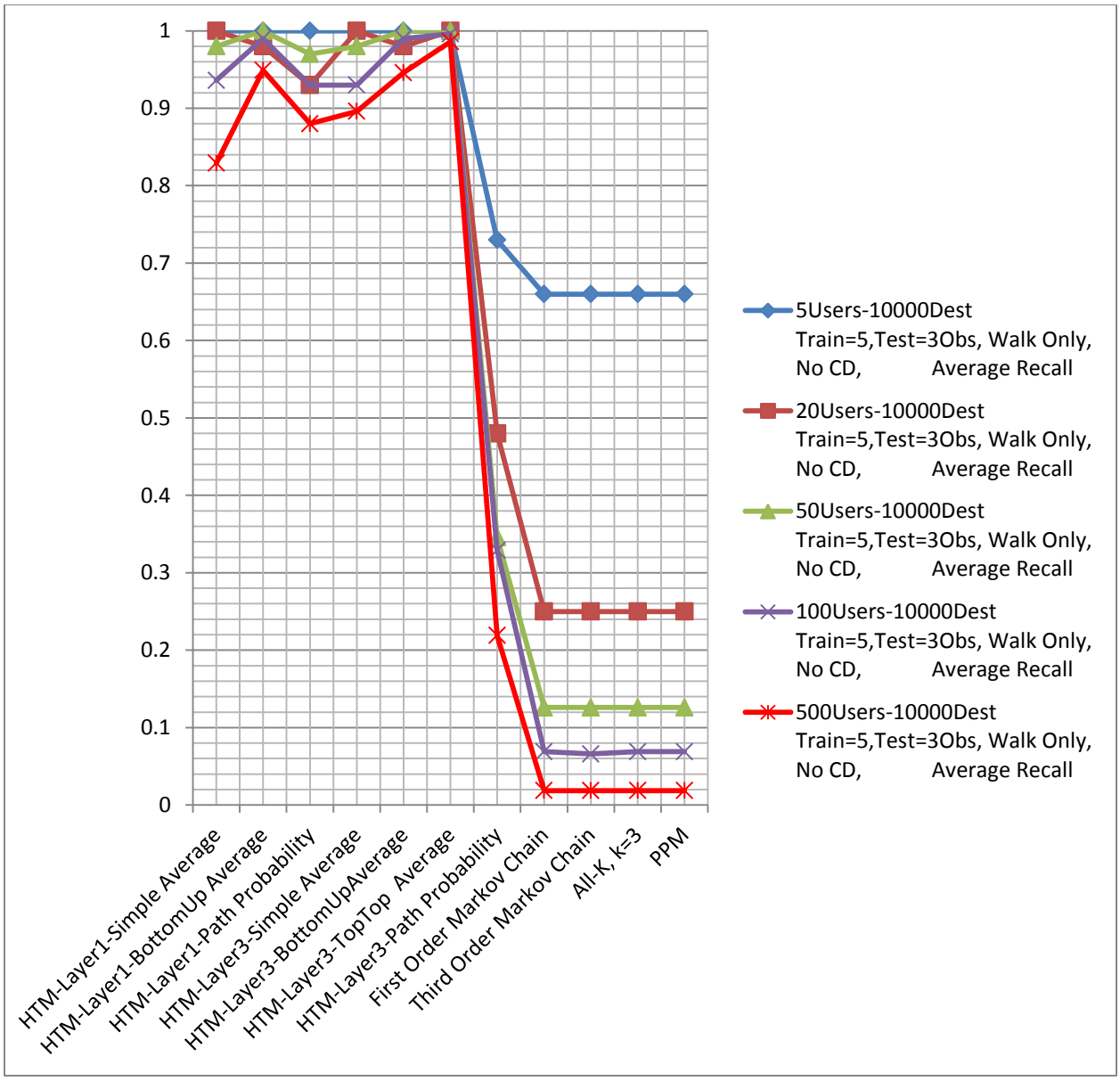


Figure 39 Experiments for 5 Train days, 3 Observations for test data, 10,000 destinations

Accuracy Scalability

Accuracy scalability, in this study, represents the ability of the solution to provide consistent levels of accuracy during training and inference when the number of users increases while keeping constant the number of destinations and vice versa increasing the number of

destinations as the number of users remains constant. It is important to distinguish accuracy scalability from run time performance scalability which instead deals with how well the solution manages computing resources (CPU and Memory utilization) to accomplish its task.

The experiments in the previous section report substantial accuracy improvements when the number of visited web destinations rises from 1000 to 5000. Further increasing the number of visited web destinations from 5000 to 10,000 brings only marginal accuracy improvements. Many experiments were run to support this conclusion (see Table 13).

A separate set of experiments was also conducted and reported in this section to explore more in depth the explicit accuracy scalability of the HTM from a user point of view. In order to address the gap in number of users from Table 13 between 100 and 500 users, experiments were run utilizing synthetic data which tested only the two best performing algorithms for each layer of the HTM: *BottomUp* Layer 1 and *TopTo*p layer3. These experiments were run with 5000 web destinations using 5 days of training and 3 test observations for the following number of users: 150, 250, 350 and 450.

Figure 40 shows that overall accuracy, measured in terms of recall and precision, is good with accuracy values remaining at or above 95% as the number of user increases up to 500 users. The HTM *TopTo*p layer3 algorithm provides the most consistent accuracy performance with values of 99% for both recall and precision as the number of users under test increases.

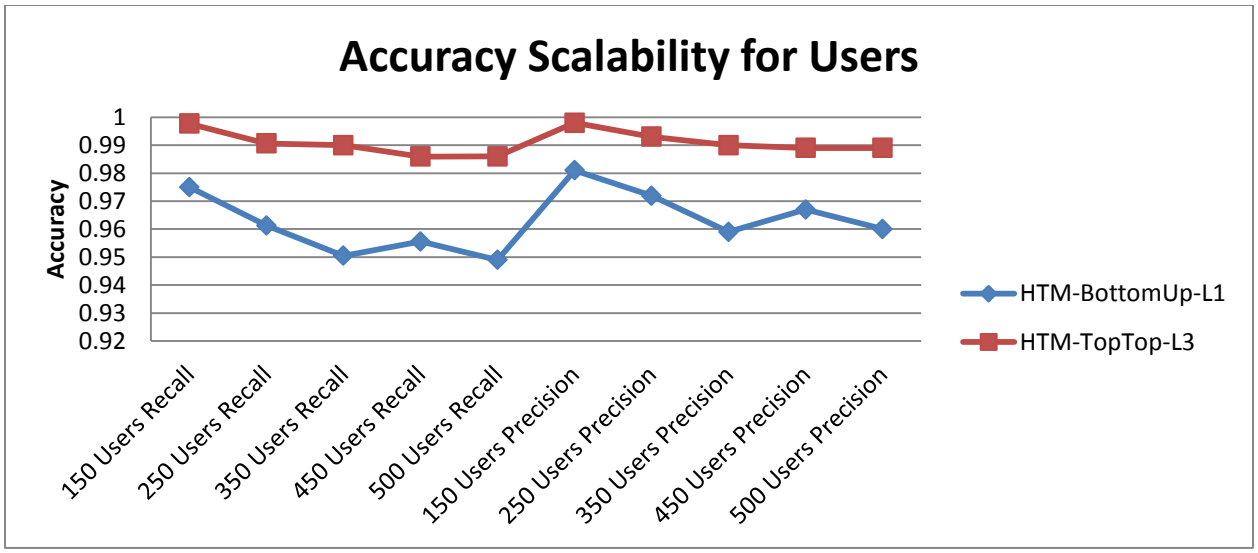


Figure 40 Accuracy Scalability 150 to 500 users using synthetic data

Figure 41 shows a different way to look at accuracy scalability by measuring the percentage change in recall and precision results when the number of users increases. When the number of users increases from 11% to 67%, the overall recall and precision percentage change values never go above 1.5%. Positive recall and precision percentage change values indicate a loss of accuracy when the number of users increases, while a negative recall and precision percentage change value indicates a gain of accuracy as the number of users increases. These results confirm that when measured against synthetic data, HTM accuracy performance for users scales well.

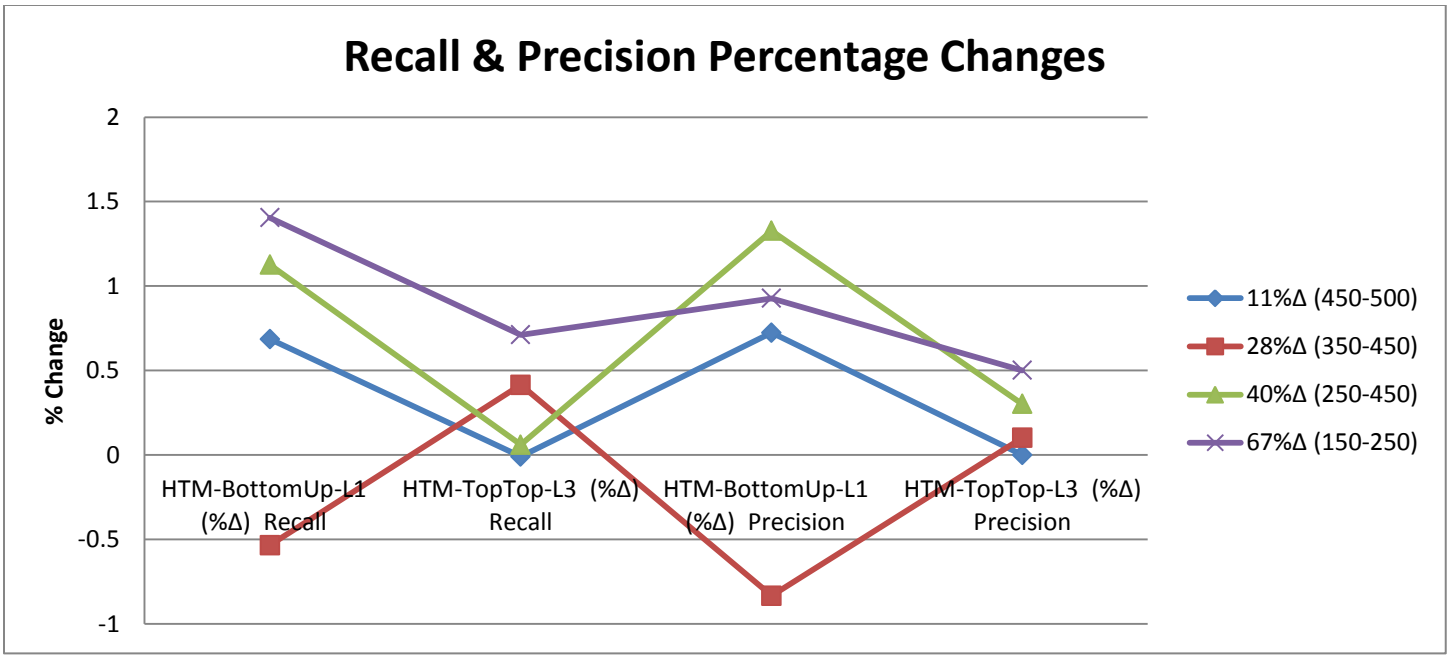


Figure 41 Recall and Precision percentage changes for accuracy scalability measurements

Results of experiments to verify user attribution accuracy with concept drift

The next set of experiments tries to measure how accuracy for the HTM and alternate algorithms is impacted by potential changes in the behavior of the user over time (concept drift). These changes are reflected in the synthetic data set and take one of two forms:

- **Random Walk** where 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 current node V_j. Details for this algorithm can be found in section "Generation of Synthetic Data for the Simulation".
- **Context Drift** in the form of either 20% new connections to already existing learned nodes or 10% new connections to new nodes not learned before.

Table 15 shows all 8 sets of experiments that were conducted using synthetic data with different forms of concept drift. The experiments were run against both HTM and alternate

algorithms so that each square in Table 15 represents 11 experiments. For these experiments the number of users and the number of destinations remained constant with the following parameters changing:

- Train Days: 5, 10, 15, 20 and Test Days: 2, 3, 4, 5
- Level of concept drift: None=Walk Only, Concept drift via: Random Walk, New Connectivity (20%) and New nodes (10%)

Number Destinations/ Number of users	5 Training Days/2 test Days	10 Training Days/3 test Days	15 Training Days/4 test Days	20 Training Days/5 test Days
1000/5 (Baseline)	Walk_Only, No CD	Walk_Only, No CD	Walk_Only, No CD	Walk_Only, No CD
1000/5	Random Walk, No CD	Random Walk, No CD	Random Walk, No CD	Random Walk, No CD
1000/5	Walk_Only, 20% Connectivity,10% New Nodes	Walk_Only, 20% Connectivity,10% New Nodes	Walk_Only, 20% Connectivity,10% New Nodes	Walk_Only, 20% Connectivity,10% New Nodes

Table 15 Accuracy tests completed using Synthetic data with concept drift

The results all experiments shown in Table 15 are reported in Appendix B. The rest of the discussion in this section only reports a key subset of the overall results from Appendix B in order to determine how well the goals of this study were met.

Figure 42 shows the impact of applying the random walk and concept drift algorithms to a base line implemented using the walk only algorithm for 1000 destinations, 5 users with 5 days of training and 2 days' worth of test data.

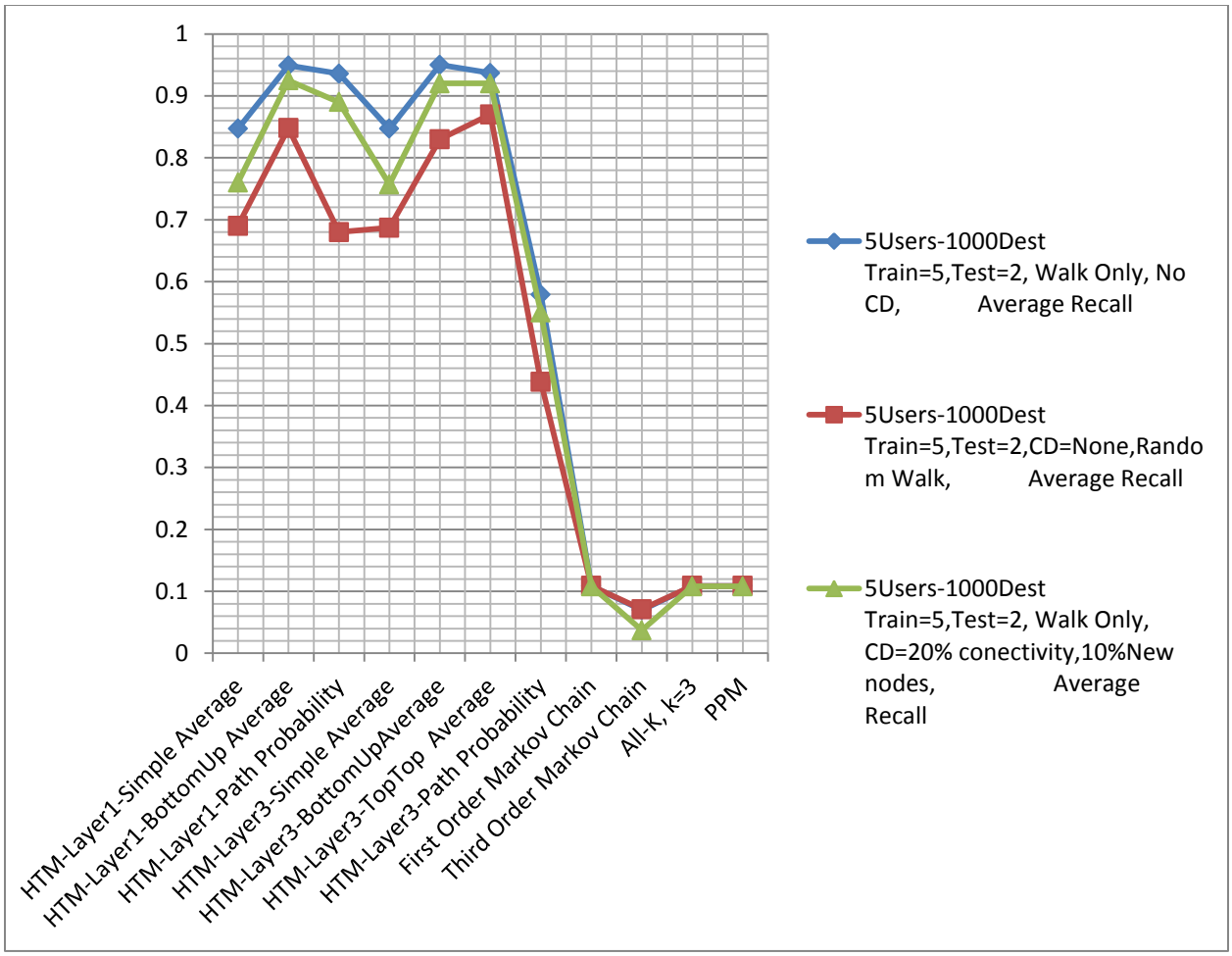


Figure 42 5 Users, 5 Train Days, 2 Test days using concept drift

The random walk algorithm applied to synthetic data impacts accuracy the most for HTM algorithms but not as much for alternate algorithms. Figure 43 represents the same data set used in Figure 42 and shows the difference from the base line for both random walk and concept drift algorithms. The further away from the zero baseline recall readings fall, the more that algorithm implementing a form of concept drift impacts the accuracy of the HTM. Appendix B shows the rest of the graphs and tables for different number of train and test days.

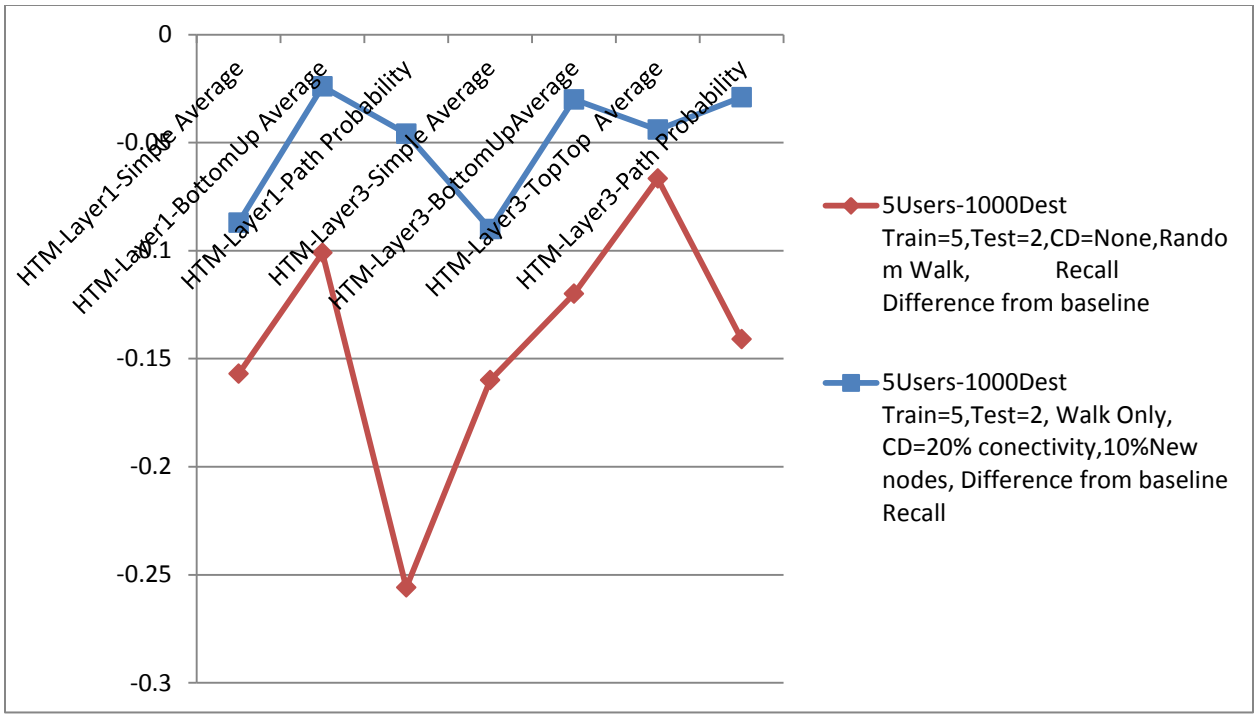


Figure 43 Difference in recall from the baseline of the "Walk Only" HTM Algorithm

The higher negative impact of the random walk algorithm might not be an obvious result, but it makes sense once one understands that an important property of the random walk algorithm is that it tends to terminate existing sequences and start new sequences any time the connectivity probability of the current node V_i does not exceed a random uniformly distributed value ***r.*** Connectivity probability represents the strength of connectivity of node V_i (measured based on frequency of access to other neighbor nodes) proportional to the number of connections emanating from node V_i. The concept drift algorithm on the other hand tends to add new connections or new nodes to existing sequences and does not split them. This means that the concept drift algorithm as run for these experiments preserves 80-90% of first part of a sequence, modifying the last 10-20%. The random walk algorithm starts new sequences where the first element of the new sequence is not selected from the trained data set but from a zipf distribution. This condition occurs more for nodes that are visited less often during the training session. The

results of experiments conducted for different train and test day combinations (shown in Appendix B) show similar results.

Results of experiments using real network data from a cellular data network

The next set of experiments used real network data collected from a CDMA cellular data network in North America over a period of approximately a month from 12-17-2012 to 1-18-2013. The data collected includes real timestamps and real destinations. TCP timestamps were not collected since they were not included in the retrieved traces and instead they were synthetically generated to support the HTM training phase of the experiments. Experiments were conducted against the HTM using the following parameters in order to provide a basis of comparison with experiments conducted with synthetic data: 5 and 10 users, 5000 destinations, 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 about 5200.

Figure 44 shows the results of the experiment using real network data for 5 users, 5 train days and 1 test day. As can be seen the HTM algorithms performed poorly when using real network train and test data compared to equivalent synthetic data.

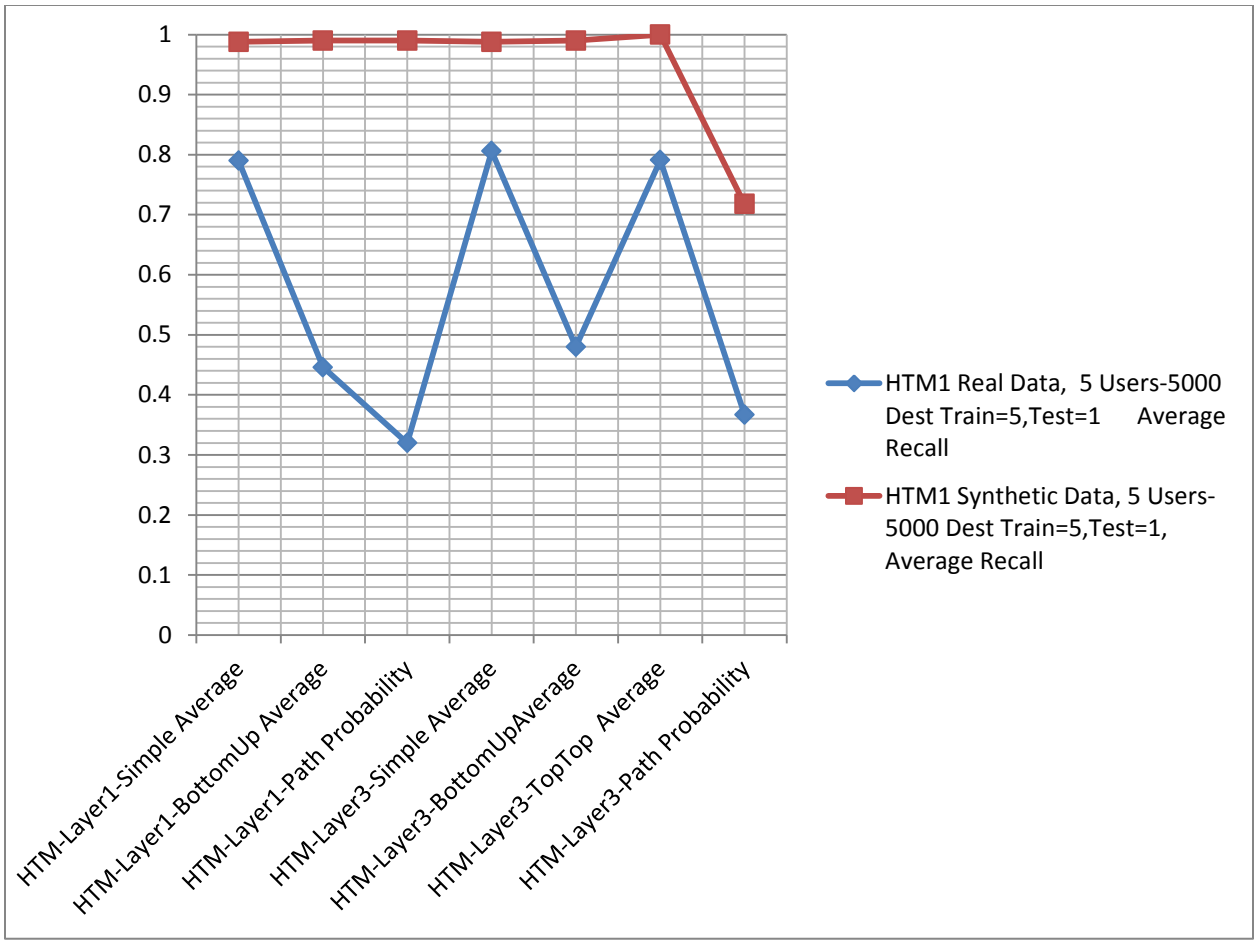


Figure 44 Real Data HTM1, 5000 destinations, 5 users, Train 5 days, Test Days = 1, Average Recall

When the HTM prototype was calibrated using real network data, as shown in Table 16, it was discovered that not only the calibration was no longer perfect (100% as was the case for synthetic data) but for some HTM algorithm the accuracy was extremely low.

Users	Layer 1 HTM Algorithms % Accuracy	Layer 3 HTM Algorithm % Accuracy
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	99% (1 error)	99% (1 error)
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	78% (48 errors)	79% (46 errors)
User-2	82% (53 errors)	85% (44 errors)
User-3	81% (22 errors)	81% (21 errors)
User-4	74% (38 errors)	76% (35 errors)
User-5	71% (4 errors)	79% (3 errors)
		Average Top-Top
User-1		100%
User-2		99% (1 error)
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	6% (206 errors)	4% (215 errors)
User-2	27% (221 errors)	22% (236 errors)
User-3	61% (44 errors)	57% (48 errors)
User-4	41% (86 errors)	32% (100 errors)
User-5	36% (9 errors)	36% (9 errors)

Table 16 HTM1 Calibration results with Real Network Data for 5 users, 5 Train Days and 2 Test days

Why is the HTM performing so poorly with real network data?

Visual observation of both train and test real network data showed a high recurrence of repeating patterns of a single destination as in: 48 48 48 48 48 48 48. This can be attributed to two main reasons:

1. Multiple visits to the same web sites occur with the same time stamp. This occurs when an individual user's web page retrieves multiple images from the same visited web site
2. Multiple visits to the same web sites occur with different time stamps showing that indeed users tend to visit the same web site repeatedly

This repeated continuous pattern is not handled well by the HTM. This is because the HTM breaks up repetitive patterns. So pattern, 1,2,3 1,2,3 1,2,3 is sees as pattern 1,2,3 occurring 3 times (which is good), but sequence 2,2,2,2,2,2,2,2 is 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, the HTM does not see a sequence of destinations but only a single element.

How did calibration of alternate algorithms perform with real network data?

Table 17 shows the results of calibration test runs for all alternate algorithms. The results of calibrations of alternate algorithms are better than equivalent results using the same data set for HTM algorithms.

Users	**Alternate Approaches** **% Accuracy**
	1st Order Markov
User-1	100%
User-2	99% (1 error)
User-3	100%
User-4	100%
User-5	
	3rd Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	All K Order Markov (K=3)
User-1	100%
User-2	99% (1 error)
User-3	99% (1 error)
User-4	100%
User-5	100%
	PPM
User-1	100%
User-2	99% (1 error)
User-3	100%
User-4	100%
User-5	

Table 17 Calibration results for Alternate Approaches using real network data for 5 users, 5 Train Days and 2 Test days

It is important to note that the performance of alternate algorithms continued to be poor against real network data in a non-calibration scenarios with a train data set of 5 days and a test data set of 2 days, as shown in Figure 45. This leads to another observation regarding the calibration process. Good calibration results, while important to qualify algorithms for experiments, do not necessarily guarantee good results with test data that differs from the train data set.

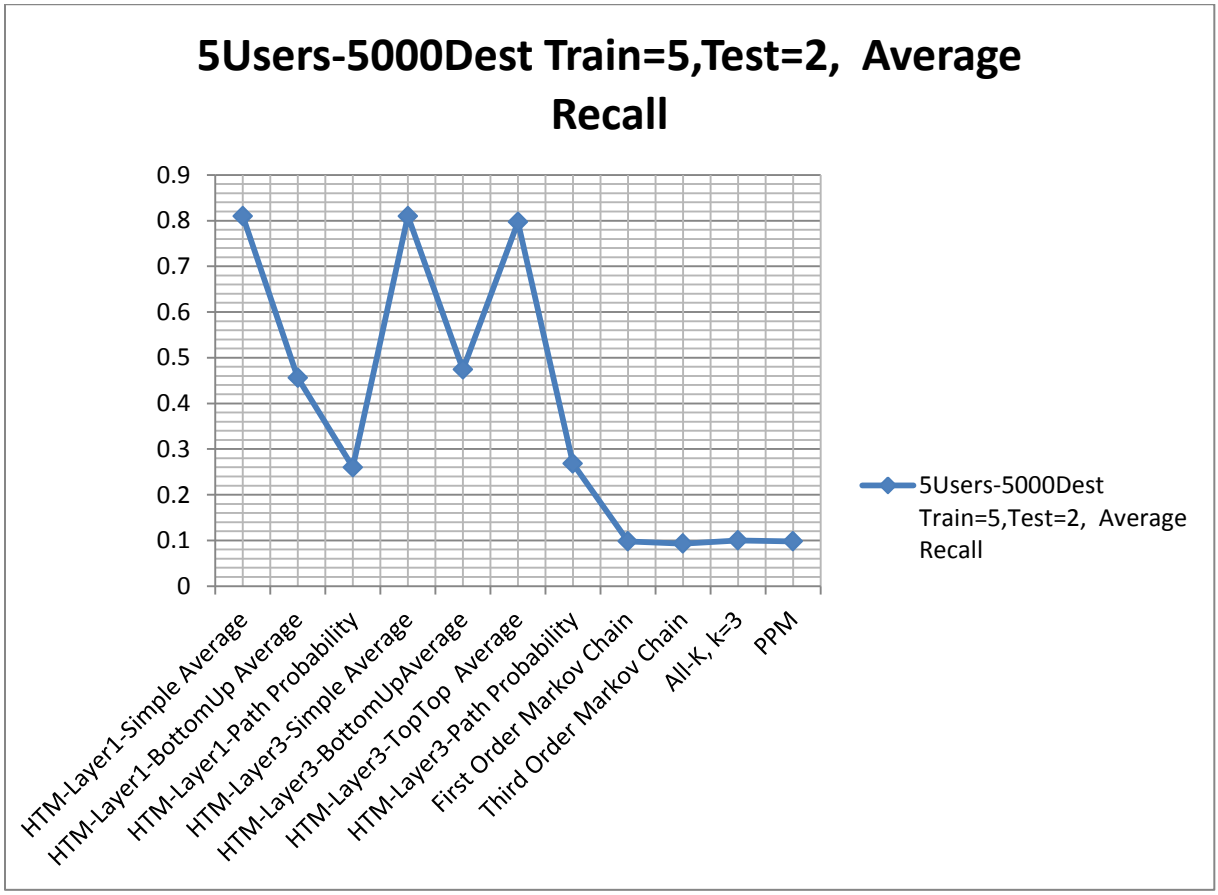


Figure 45 HTM1 Real Network Data comparison with Alternate Algorithms

The original HTM (HTM1) was modified into a new version HTM2 which accounted for multiple repeated destinations at layer 1. HTM2 was calibrated and reported the accuracy shown in Table 18. The accuracy reported is still below the 100% level of performance achieved with

synthetic data but performance improved from the previous calibration results for bottom up and path probability algorithms.

Users	Layer 1 HTM Algorithms % Accuracy	Layer 3 HTM Algorithm % Accuracy
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	99% (1 error)	100%
User-3	100%	99% (1 error)
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	99% (1 error)	99% (1 error)
User-2	99% (1 error)	99% (1 error)
User-3	98% (2 errors)	98% (2 errors)
User-4	100%	100%
User-5	100%	100%
		Average Top-Top
User-1		100%
User-2		99% (1 error)
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	100%	93% (14 errors)
User-2	99% (3 errors)	96% (12 errors)
User-3	100%	99% (1 error)
User-4	100%	97% (4 errors)
User-5	100%	100%

Table 18 HTM2 Calibration results with Real Network Data for 5 users, 5 Train Days and 2 Test days

Figure 46 shows the results of running HTM2 against the original synthetic data set (baseline) and the real network data sets. The performance of HTM2 improved over the previous version HTM1 for all algorithms, yet it still lags behind the performance of the baseline synthetic data.

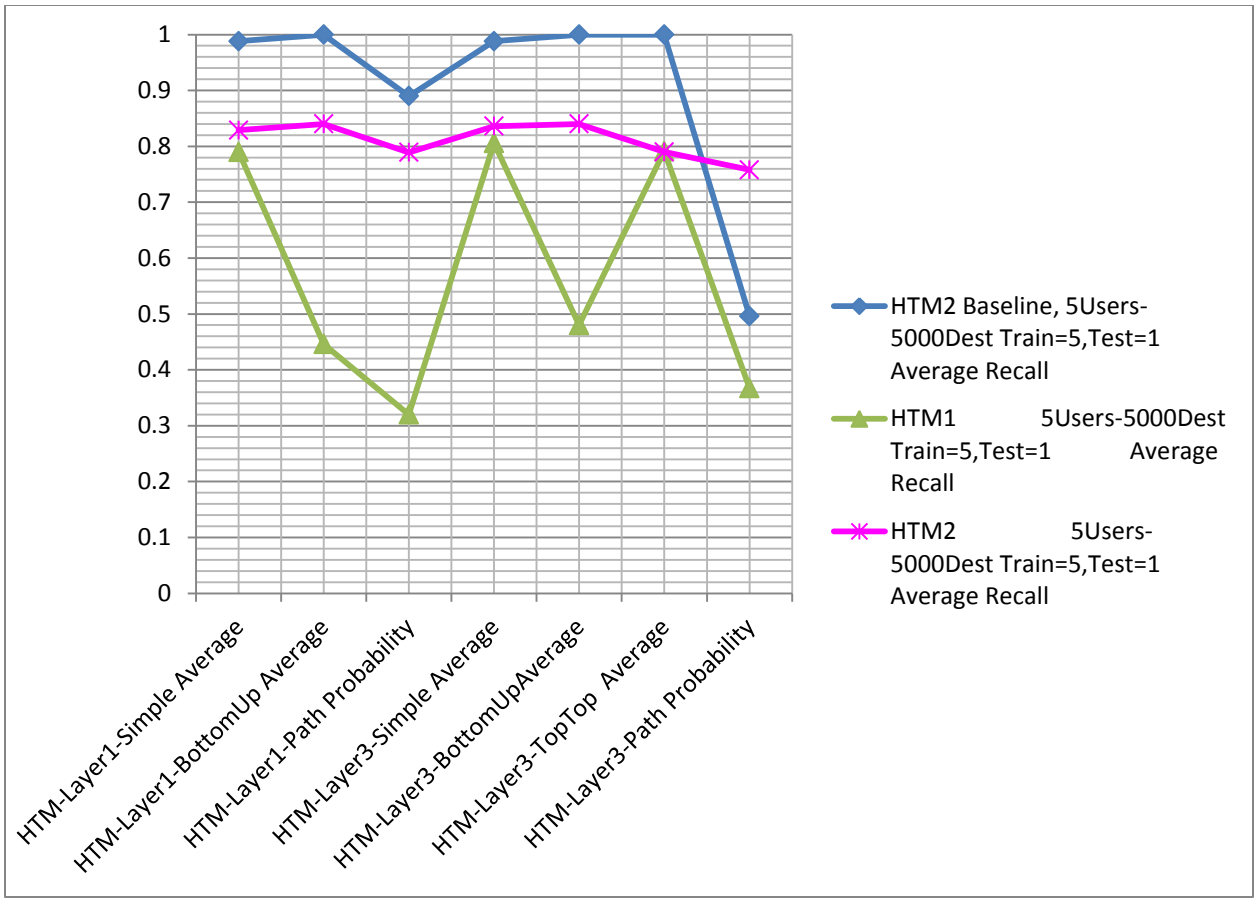


Figure 46 Real Data HTM2, 5000 destinations, 5 users, Train 5 days, Test days = 1, Average Recall

The repetitiveness of the certain destinations within observations was believed to impact the performance accuracy of the HTM. In order to better understand how this property of real network data impacts the experiments, an intra-observation repetitiveness statistic (see Appendix C for the MATLAB script) was created and applied to the real network data set for 5 users, 5 train days and 2 test days. Intra-observation repetitiveness statistic measures uniqueness of destinations within an observation. For instance, a 75% value for this statistic given input [3 3 3 3] means that three in four destination in the input repeat. If the input was [1 1 2 2 3] then intra-observation repetitiveness would be 40% or two in five observations repeat. Figure 47 shows that intra-observation repetitiveness is very high for train real network data and high for test real

network. For synthetic data intra-observation repetitiveness is fairly high for the train data set but low for the test data set.

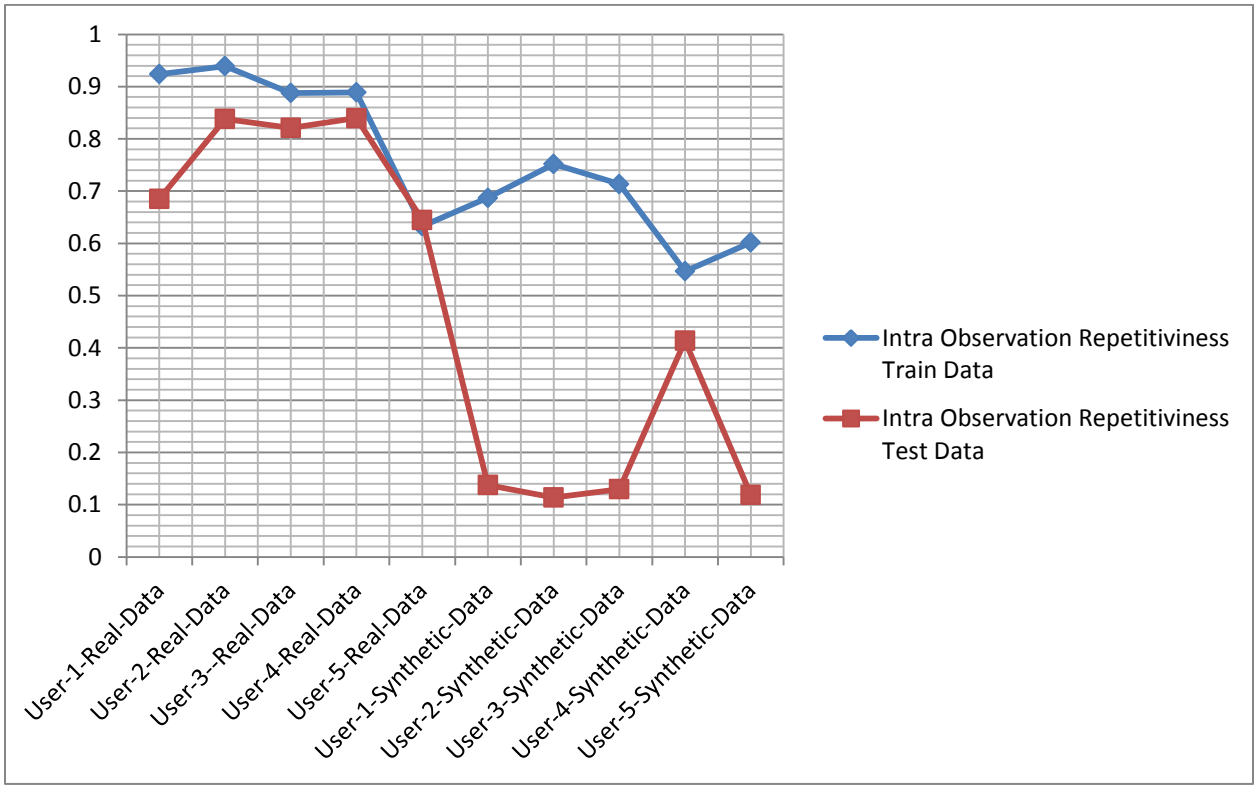


Figure 47 Intra Observation Repetitiveness statistics for real network data

In order to gain more insight into the intra observation repetitiveness results the third observation for user 3 from the real test dataset was extracted and is shown below:

48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,52,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,499,48,48

The intra-observation repetitiveness for this observation is 94 % with 47 repeating destinations out of 50. If indeed the high level of repeated destinations within observations in the data sets contributes to lower accuracy performance then reducing it should provide measurable improvements. The next set of experiments, were designed to address this question by reducing repeated destinations within observations and removing from the real network datasets any

repeated destinations that occurred at the *exact same time*, as measured in the real network data by the value of time stamps. This change does not impact the validity of the results since what is being removed from the data set includes repeated retrievals of objects like pictures belonging to a given web page. The reduction was applied to all train and test data files and accounted for a total reduction in repeated destinations of about 35%. Figure 48 shows the results running the HTM algorithm for 5 users, 5 train days 1 test day over a real network data set with reduced repeated destinations. The results show a definite improvement (HTM2++ represents the HTM runs where destination repetitiveness was reduced) but the results are still below the accuracy results of the synthetic baseline.

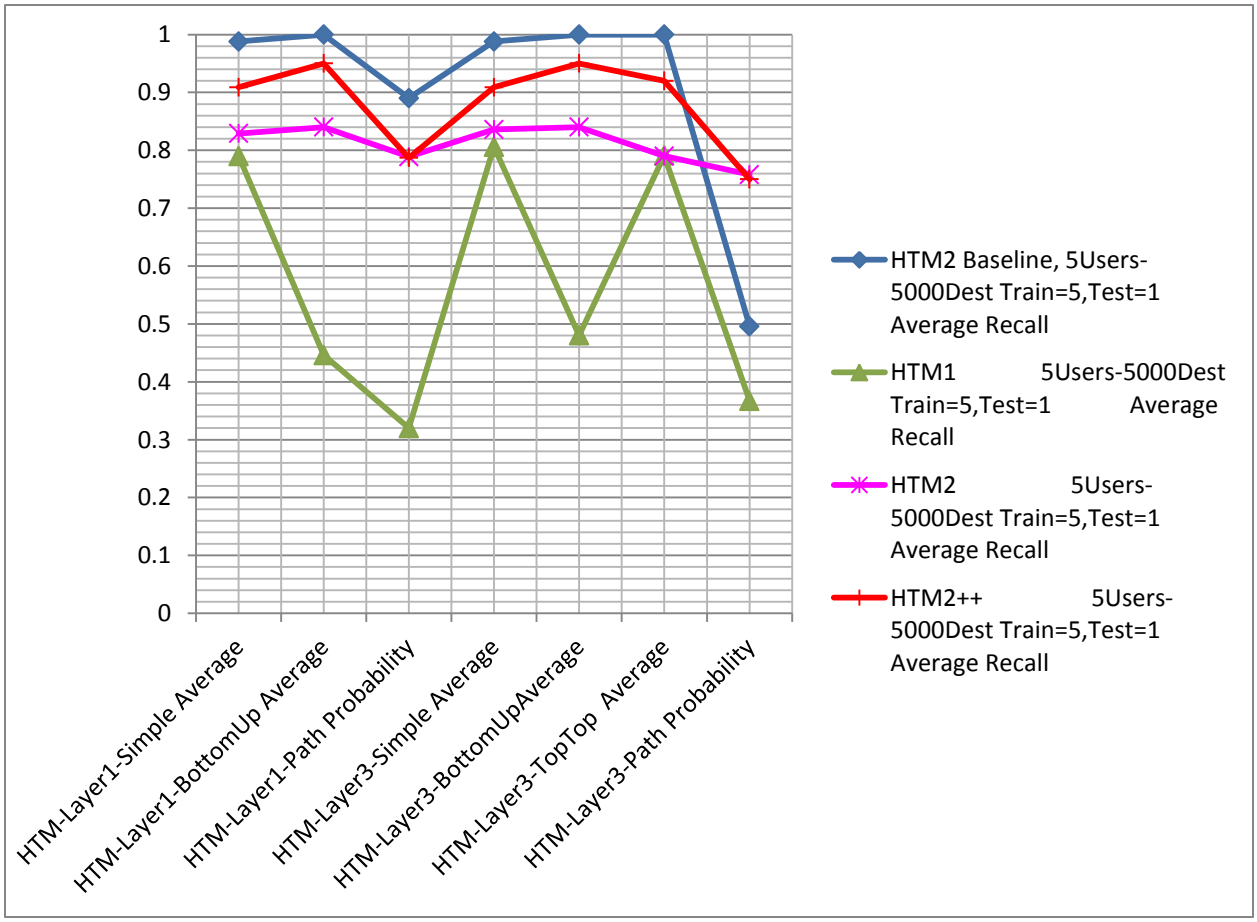


Figure 48 Accuracy comparisons of all HTM versions including removal of same time destinations 1 Test day

In order better understand the impact of the 35% repetitiveness reduction impact, the experiment was run again against 5 users, 5 train days and this time 2 test days. Figure 49 shows that while an improvement is achieved (see HTM2++) over not applying the 35% reduction, results do not go above 90% accuracy.

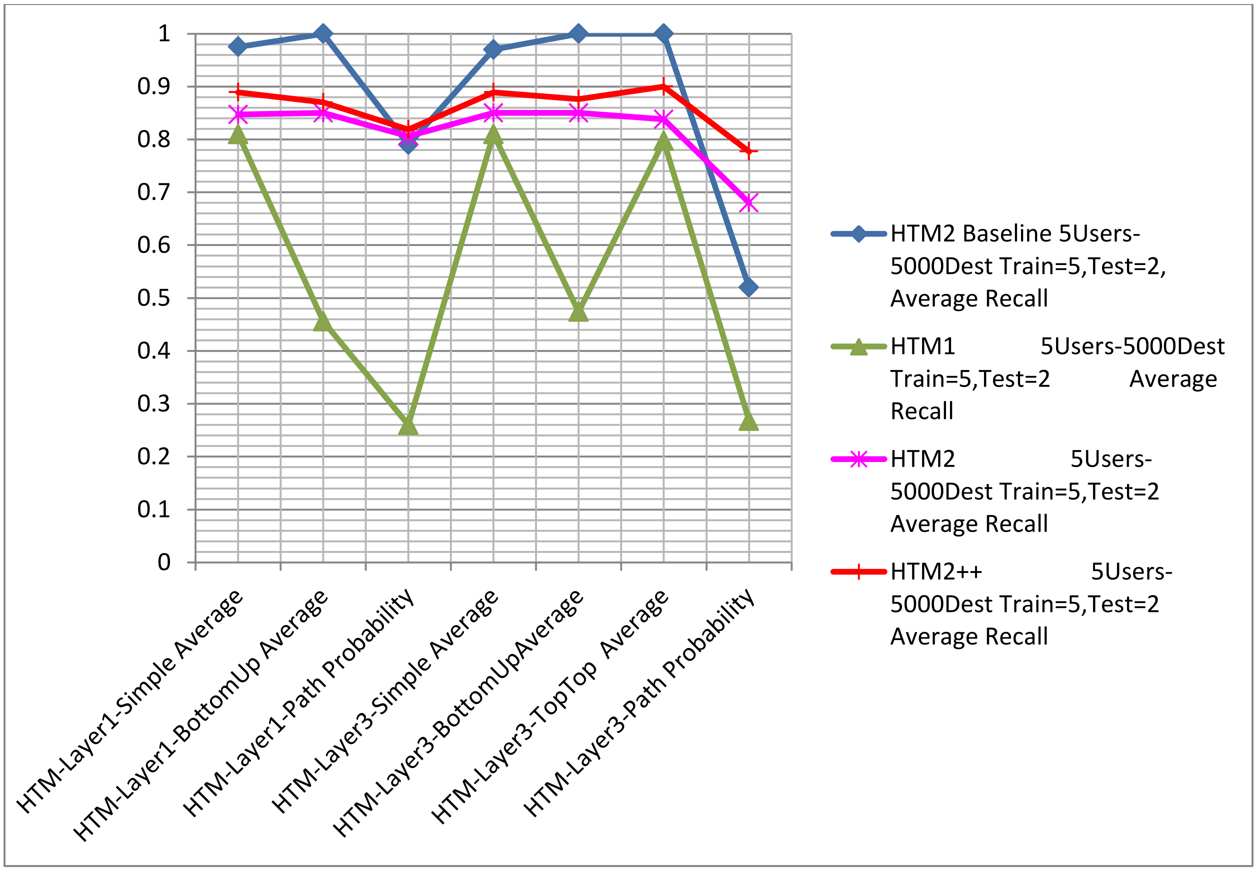


Figure 49 Accuracy comparisons of all HTM versions including removal of same time destinations 2 Test days

How did this 35% reduction in repeated destinations impact the composition of train and test data sets?

Consider the same third observation for user 3 extracted from the real test data set:

48,48,48,48,48,48,48,48,48,499,48,48,48,48,48,48,48,48,48,149,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,149,48,48,48,48,48,48,48,48,48,48,48,48,76

As shown, the observation composition of repeated web destinations was only very slightly reduced thus accounting for the improved accuracy reported by the HTM (HTM2++). The measure of inter observation repetitiveness was also implemented (see Appendix D for the MATLAB algorithm) to measure repetitiveness across observations. Figure 50 shows both intra and inter observation repetitiveness for synthetic, real network data and real network data filtered for repeated destinations within the same timestamp value. The metrics reported in this graph are averages across 5 users, with data sets of 5 train days and 2 test days. Figure 50 brings to bear an interesting inverse relationship between intra and inter observation repetitiveness, such that intra observation repetitiveness is highest for real network data and lowest for synthetic data. On the other hand, inter observation repetitiveness is highest for synthetic data and lowest for real network data, albeit the difference for inter observation repetitiveness between synthetic and real data is smaller than the difference for intra observation repetitiveness between the same data sets.

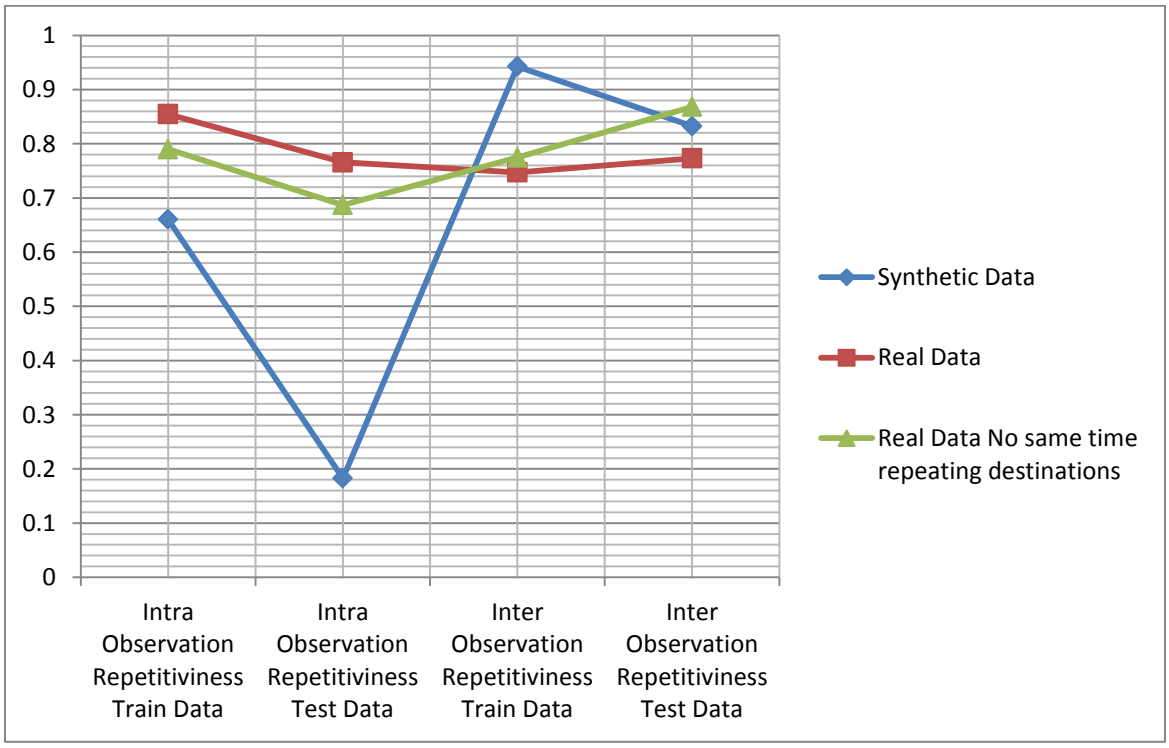


Figure 50 Comparison of Inter and Intra repetitiveness statistics for 5 users

Figure 51 and Figure 52 show accuracy results after running HTM2 on data sets with reduced repeated same time destinations. The experiments were conducted for 5 and 10 users respectively over 5 training days and 1, 2 3 test days. Accuracy performance worsens as the number of users increases from 5 to 10. Improving HTM accuracy performance for real network data that displays such extreme levels of intra observation repetitiveness is a very important topic for further study.

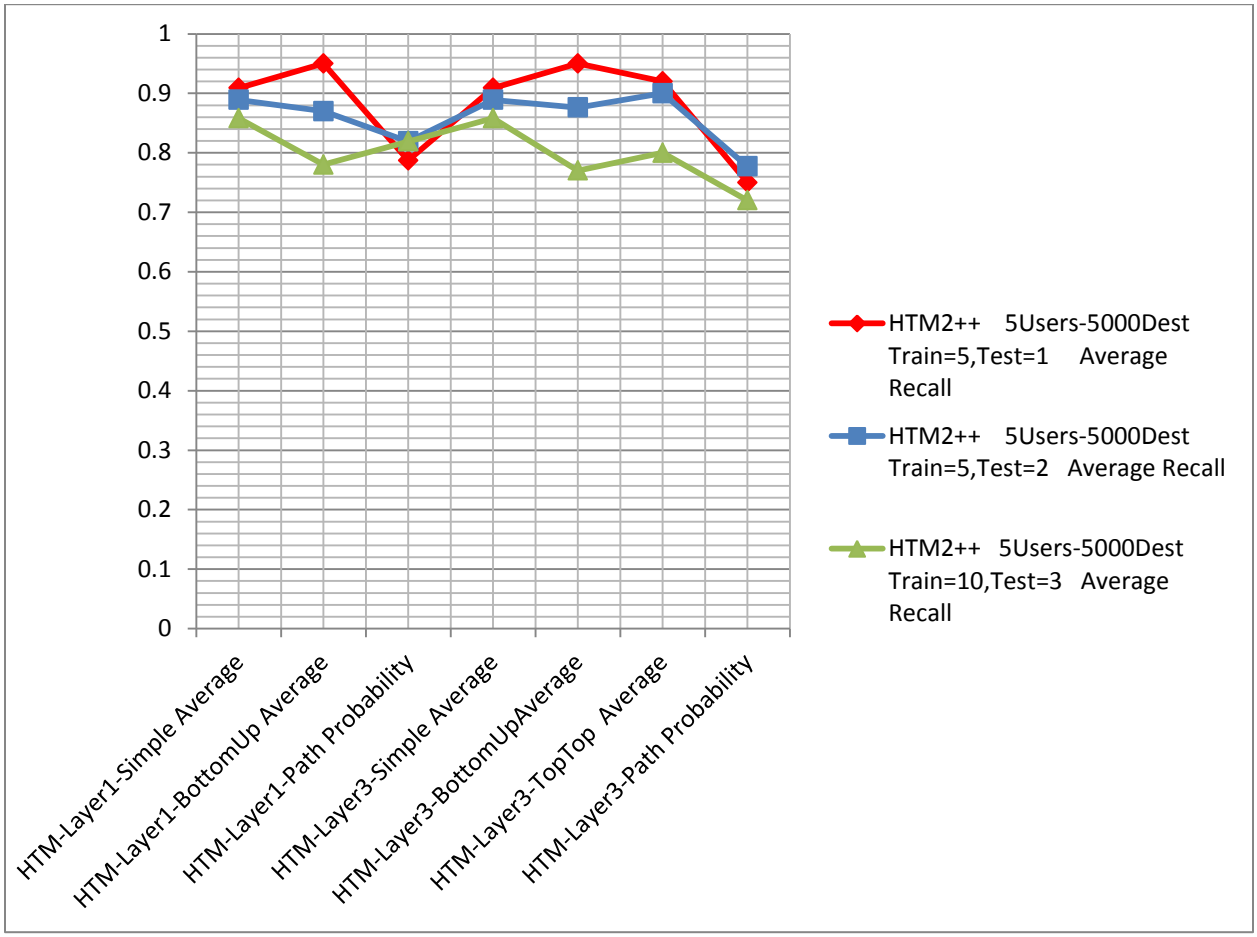


Figure 51 HTM2++ accuracy performance with real network data for 5 Users for 1, 2, 3 test days

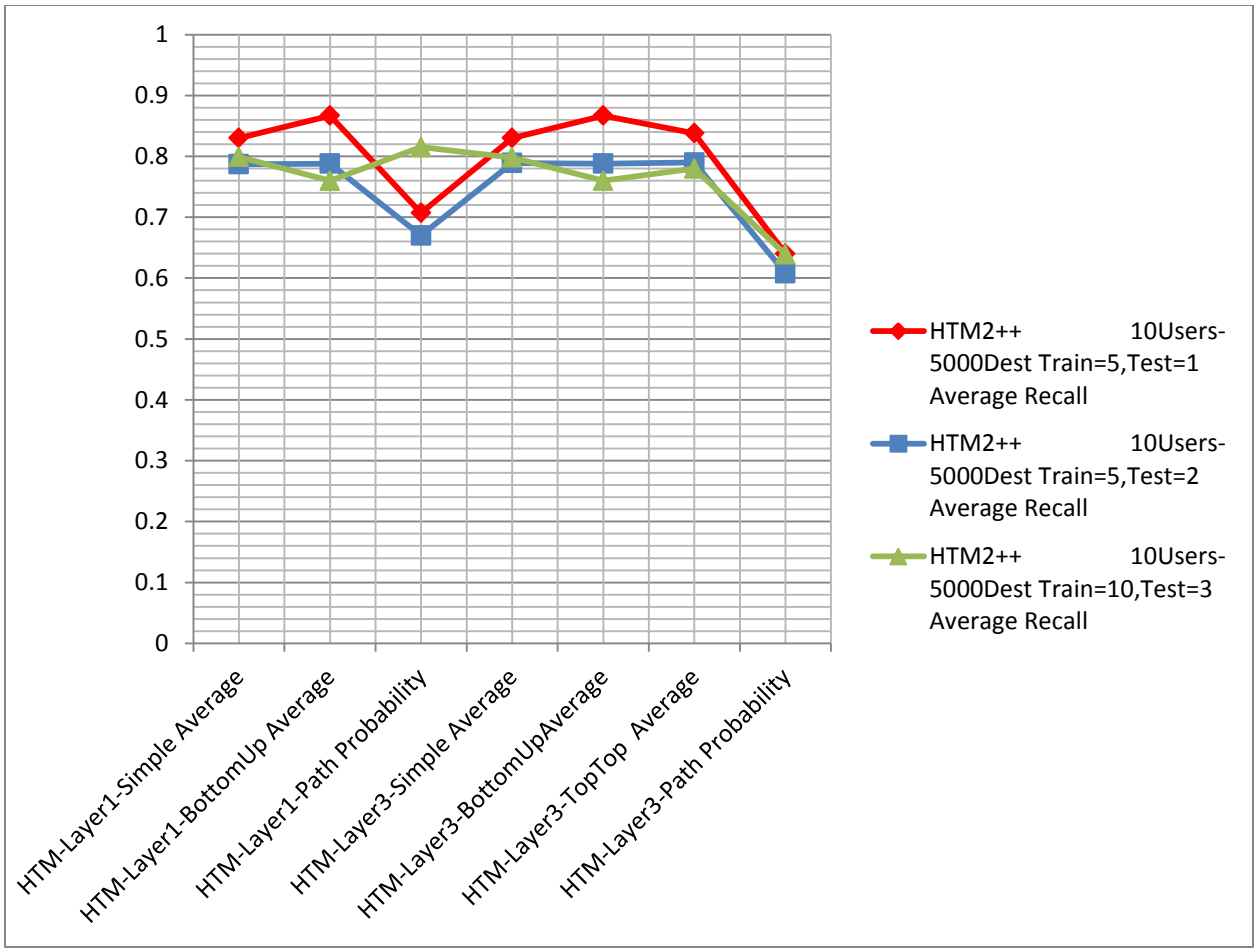


Figure 52 HTM2++ accuracy performance with real network data for 10 Users for 1, 2, 3 test days

For a complete set of statistics on HTM2 experiments using real network data see Appendix G.

Results of experiments simulating DOS Attacks

Another set of experiments run against the HTM was conducted by simulating denial of service attacks 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 start DOS attacks during the test phase. Table 19 shows all DOS attack experiments run against HTM2 (HTM version2) algorithms with all results reported in Appendix E.

Number Users/Number infected/Data Source	Number Destinations/Unit of time, Repeats every	Number Destinations/Unit of time, Repeats every	Number Destinations/Unit of time, Repeats every
10/4/Synthetic	5/5ms,5ms	10/10ms,5ms	20/20ms,5ms
10/4/Real Network	5/5ms,5ms	10/10ms,5ms	20/20ms,5ms

Table 19 Accuracy tests which simulated DOS attacks

Figure 53 shows the difference in accuracy between the recall values after and before an attack. A negative difference between the two values indicates that the HTM2 recall decreased by that value after the attack. As expected most recall differences values are negative with the difference value increasing as the number of destinations attacked increases. The BottomUp and TopTop HTM algorithm are the least impacted by the DOS attacks while continuing to outperform other algorithms (see Appendix E for more statistics on DOS attacks). Also note that while path probability is not as impacted by the DOS attacks, its performance continues to be the worst among the HTM algorithms (see Appendix E).

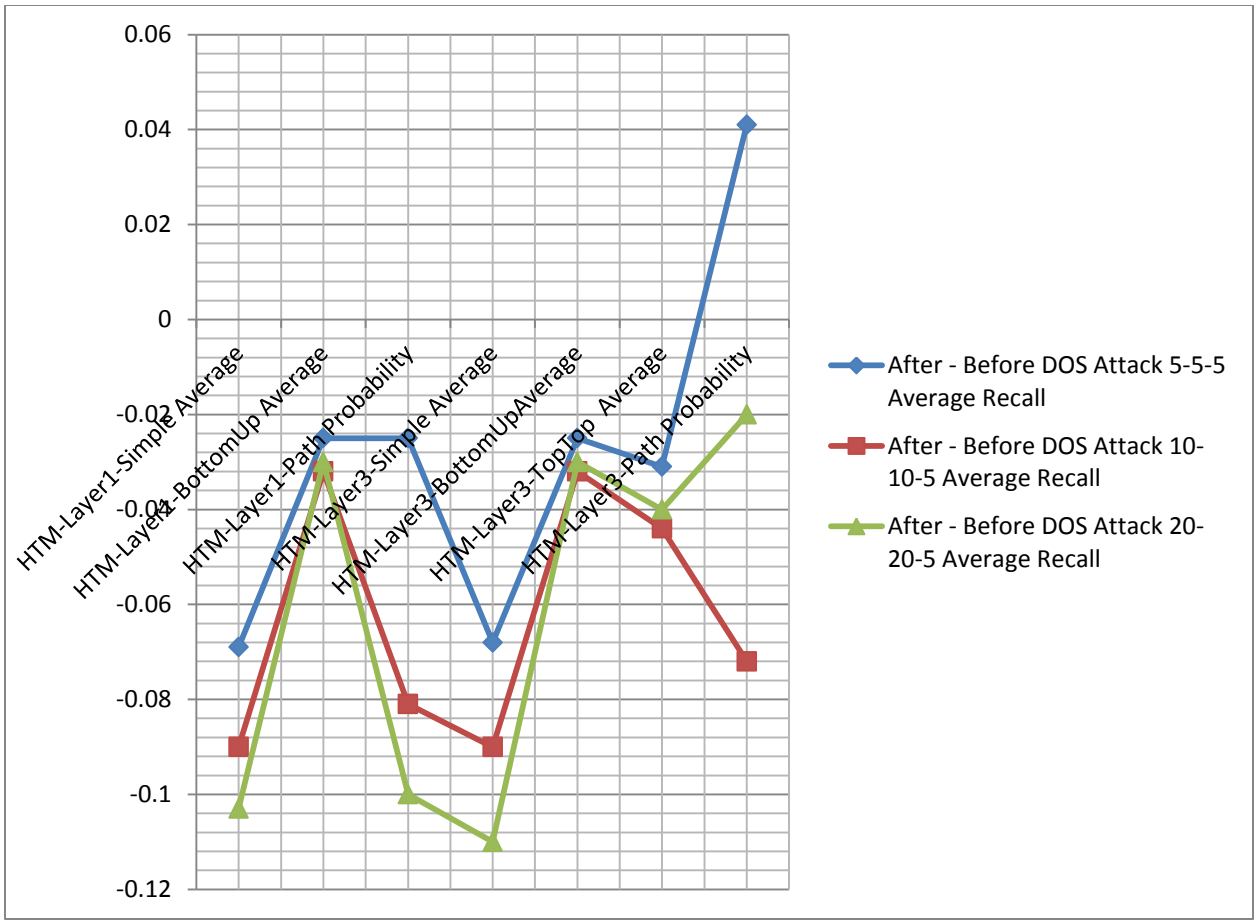


Figure 53 DOS Attack using Synthetic data against 10 users with 4 infected users

The same DOS attack experiment was also conducted with real network data as shown in Figure 54. HTM (V2) algorithms are minimally impacted (about 8%) by DOS attacks with the exception of the path probability algorithm at layer 3. In the experiments run with synthetic data (Figure 53) the maximum impact of DOS attacks did not exceed 11%.

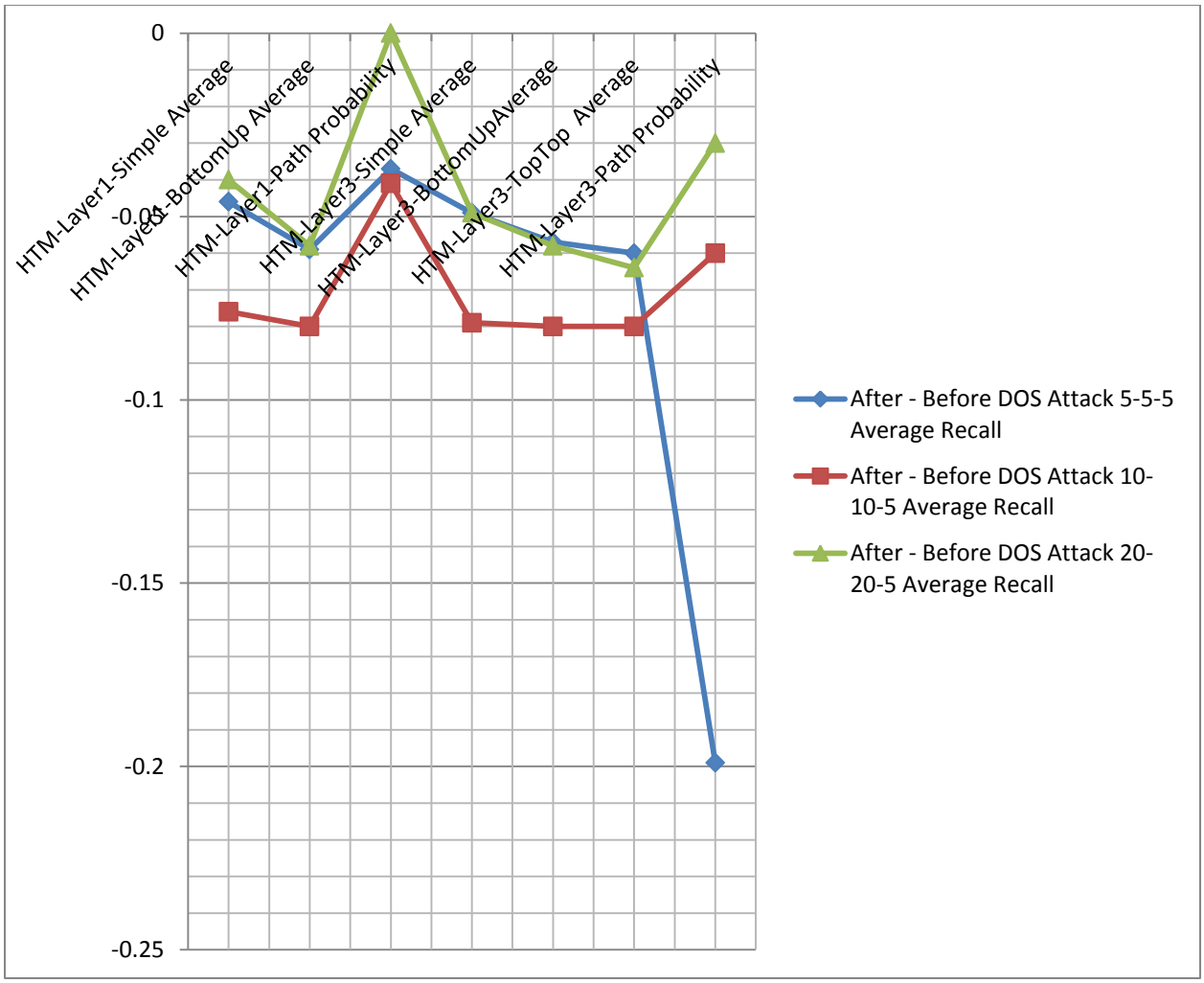


Figure 54 DOS Attack using real network data against 10 users with 4 infected users

Results of experiments simulating Phish Attacks

Another set of experiments simulates phishing attacks. 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 device 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]. Table 20 shows all Phish attack experiments run against HTM2 algorithms with all results reported in Appendix E.

Number Users/Number infected/Data Source	Number Destinations/Unit of time, Repeats every	Number Destinations/Unit of time, Repeats every	Number Destinations/Unit of time, Repeats every
10/4/Synthetic	1/1ms,[1min – 1 hour]	3/3ms, [1min – 1 hour]	5/5ms ,[1min – 1 hour]
10/4/Real Network	1/1ms,[1min – 1 hour]	3/3ms, [1min – 1 hour]	5/5ms,[1min – 1 hour]

Table 20 Accuracy tests which simulated Phish attacks

Figure 55 and Figure 56 both show that the accuracy of HTM (V2) is minimally impacted by Phish attacks, even less than for DOS attacks since accuracy drops by no more than 5% after these attacks.

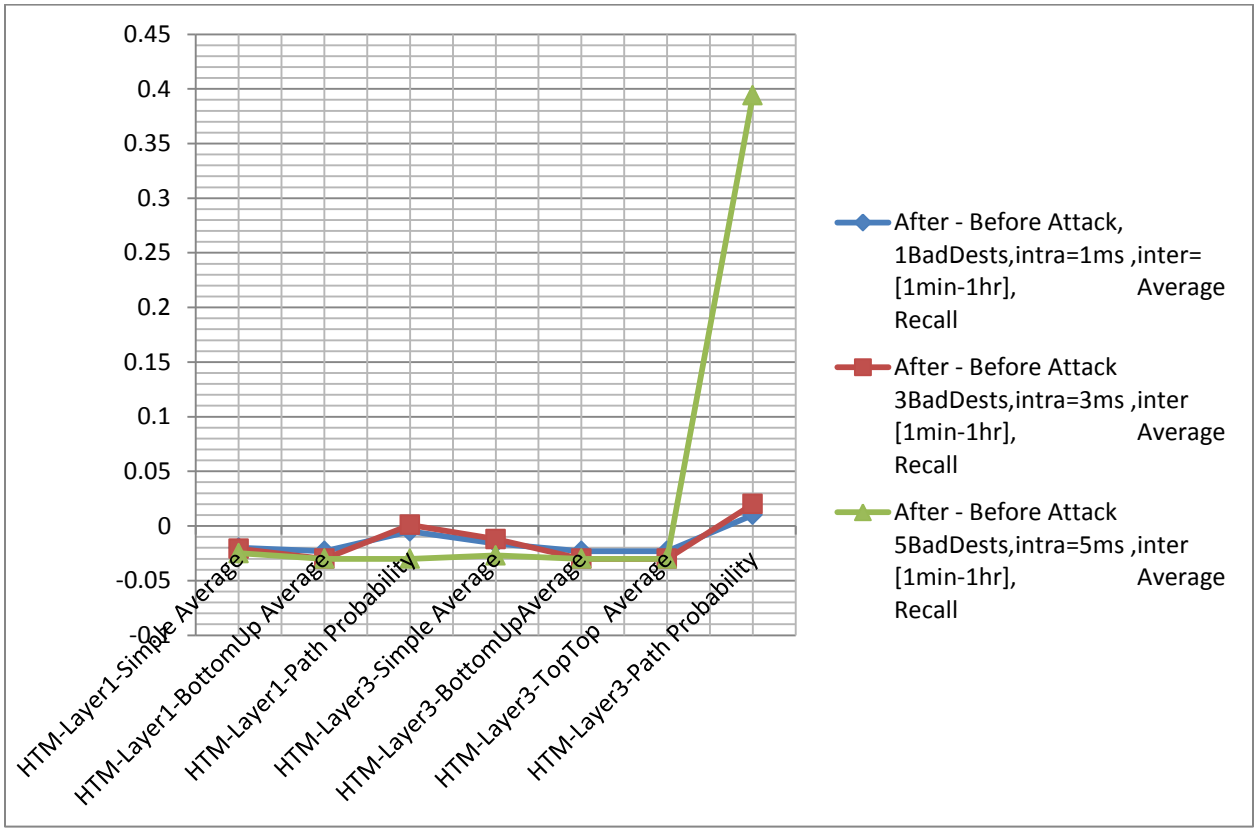


Figure 55 Phish Attack using Synthetic data against 10 users with 4 infected users

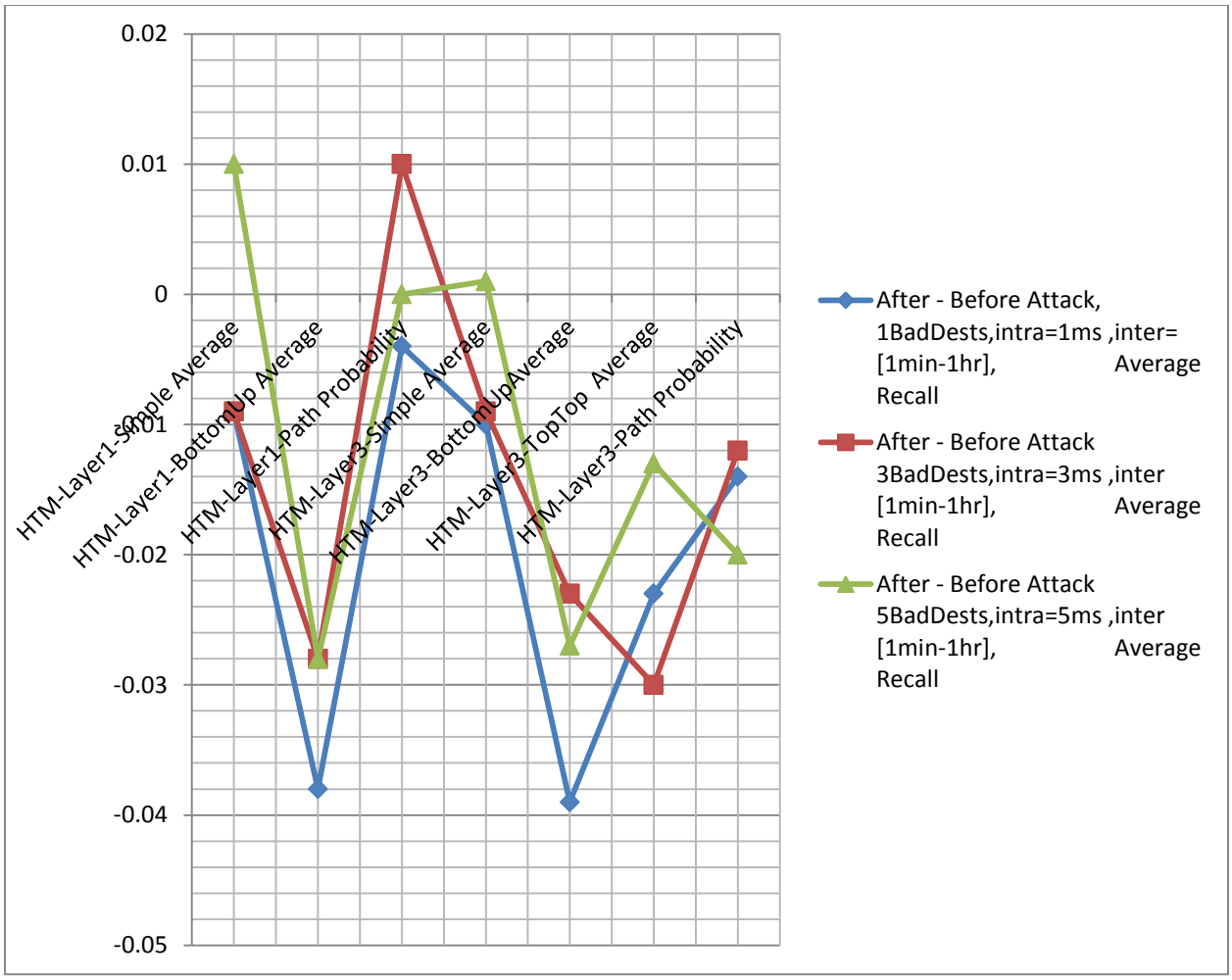


Figure 56 Phish Attack using real network data against 10 users with 4 infected users

Results of experiments for Session Identification Algorithms

The last set of experiments for this study attempts to evaluate the effectiveness of the TPC Timestamp algorithm as a session identification technique. This approach is compared to two other session identification approaches; one leveraging the source IP address and the other leveraging a sliding time window to identify sessions belonging to a specific user.

For this research, the TCP timestamp was the session identification algorithm used during the training phase, during the test phase the session identification algorithm was not used and

instead inference relied entirely on the variable Markov graphs/chains at each layer of the HTM leveraging different HTM algorithms to combine degree of membership results across observations. The idea of these 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 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 is based on real network data. The experiments include a preliminary step which runs the session identification algorithms against real network data to produce a train data set that is altered based on the bias introduce by each session identification algorithm. 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.

The session identification algorithms are:

- **Source IP**: All input with the same source IP address belongs to the same user
- **Sliding Time Window**: Select 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 a new user (source IP address) is selected at random based on users who have data falling within the sliding time window.
- **TCP Timestamp**: Uses the TCP Time stamp value within a clock skew window to track different users

The source IP and TCP Time stamps leveraged real life scenarios to alter the original train data set in the form of:

- Source IP: Source IP recycling of the same source IP address among users as done by NATs and Proxies devices
- Source IP: Re-attach of a device with new source IP as done when users move across networks
- TCP Timestamp: Data loss simulate creation of holes in the data stream
- TCP Timestamp: Device power on/off simulates resetting the TCP Timestamp

Note that the sliding window approach presented in other related literature (Banse, Herrmann, & Federrath (2012) and Yang (2010)) 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. It is important to note that under perfect conditions of no noise or device resets, no clock skew, no forced change of source IP addresses then both source IP address and TCP Timestamps are "*perfect*" tracking algorithms. On the other hand, the sliding window approach is not perfect even though it actually extends the source IP address algorithm. This is due to the fact that each time data for user-x within a sliding window runs out, a new random user is picked. For this reason no noise is added to experiments tied to the sliding window algorithm.

The number of users in these experiments is 5 and 10 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

The experiments for 5 users were run 3 times with HTM (V2) and averages over the runs computed as shown in Figure 57. The sliding window algorithm maintains the best performance even when up to 35% of the original train data set is lost as shown in Figure 58.

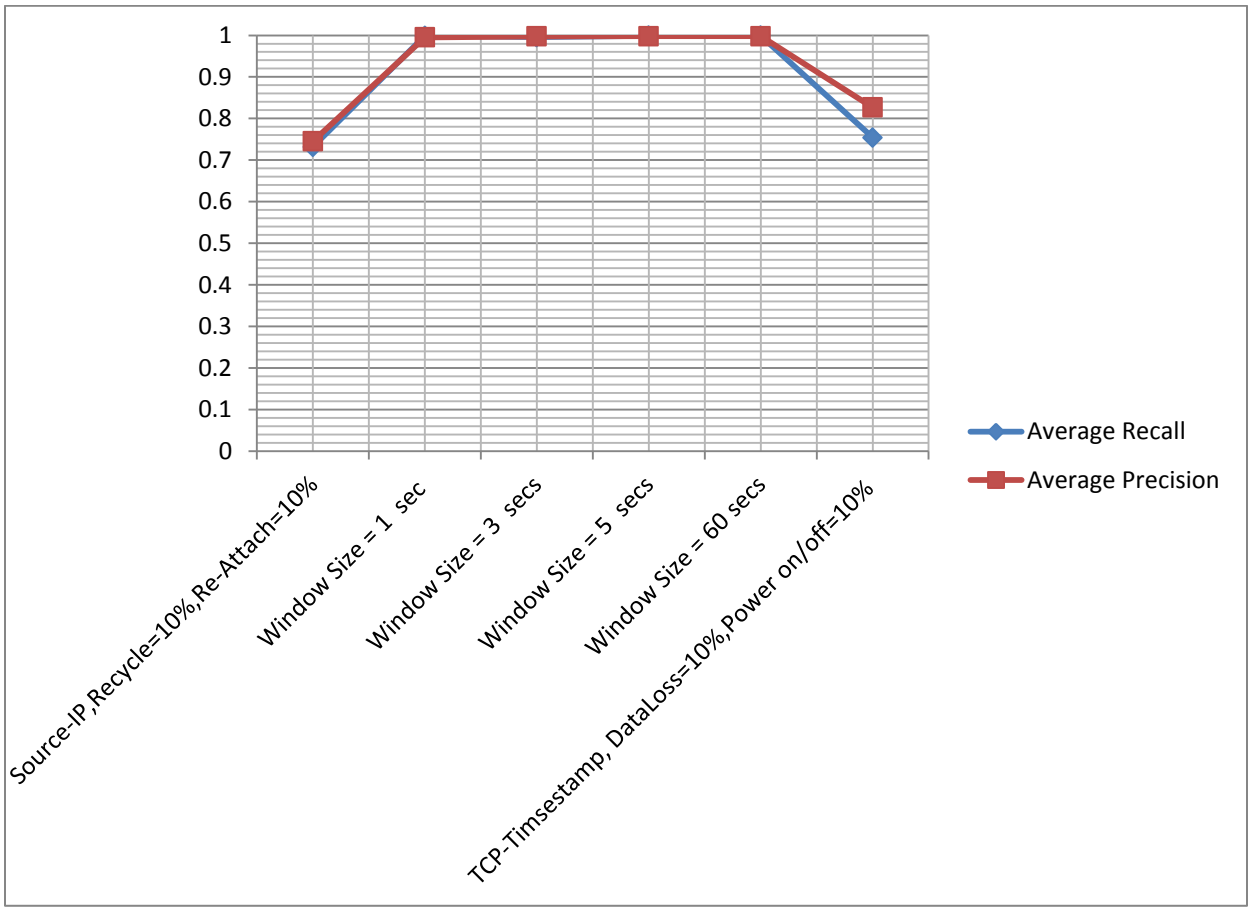


Figure 57 Accuracy of different session identification algorithms for 5 users

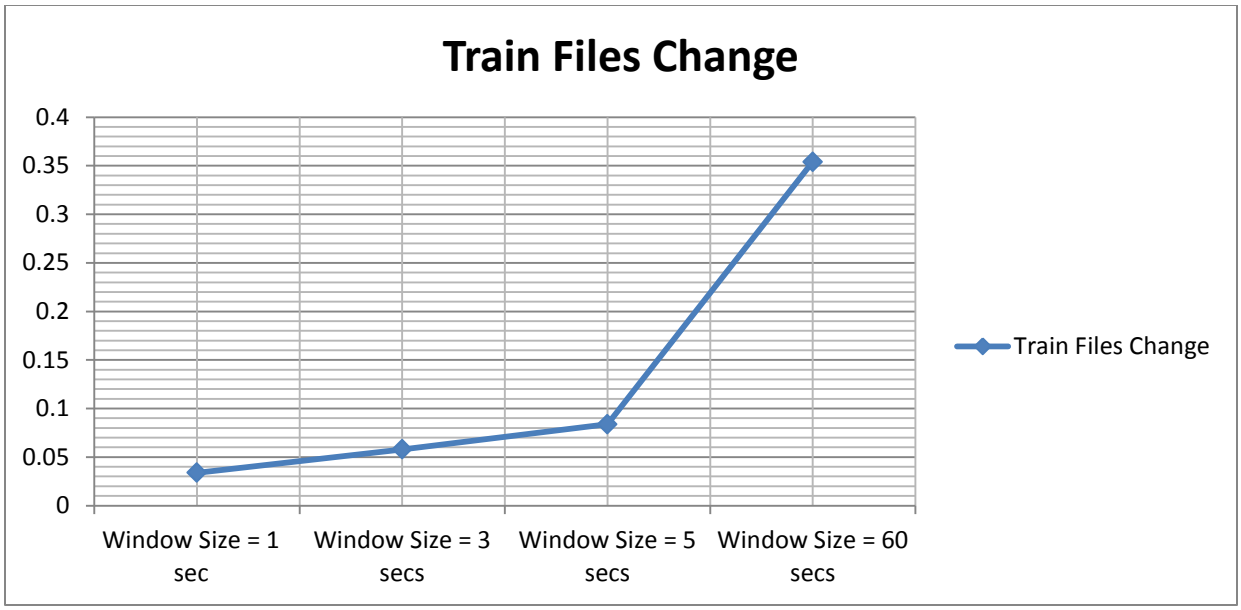


Figure 58 Percentage change for 5 users to train files resulting by use of window algorithm

For 10 users the sliding window algorithm continues to outperform the other session identification algorithms as shown in Figure 59. Figure 60 shows that even with a window size of one minute and a 57% of the original train data set lost, accuracy is still quite good at 94%.

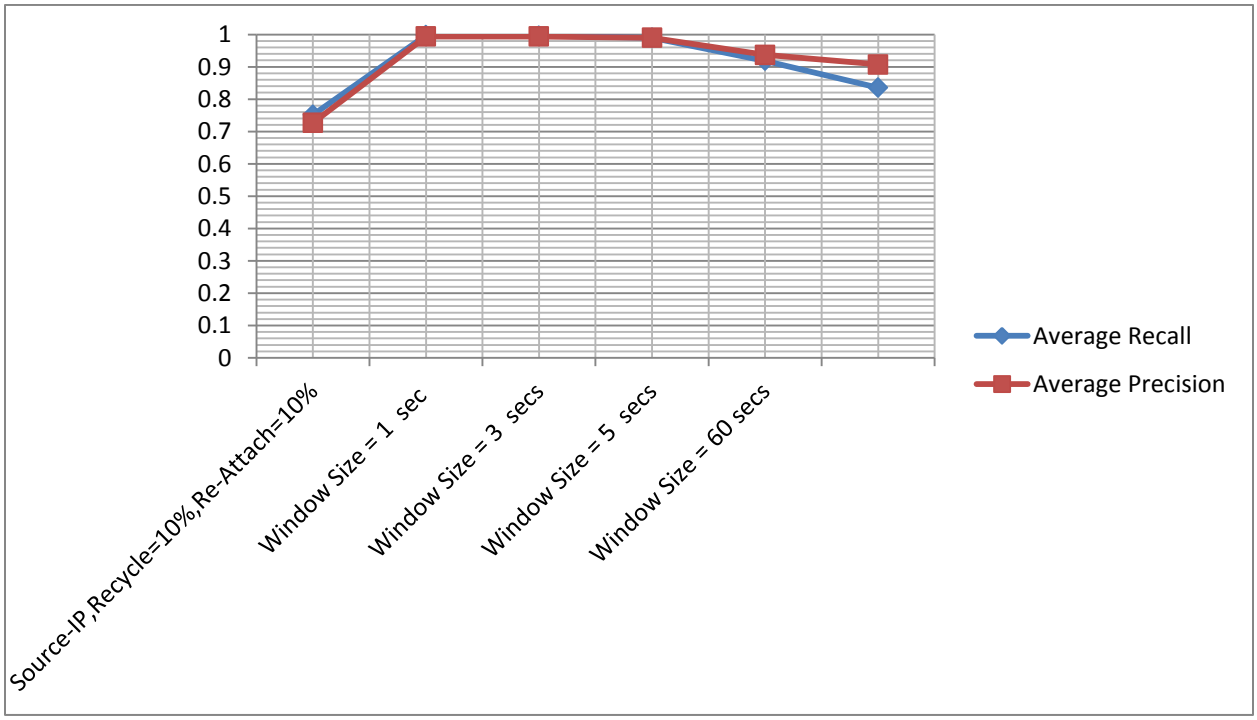


Figure 59 Accuracy of different session identification algorithms for 10 users

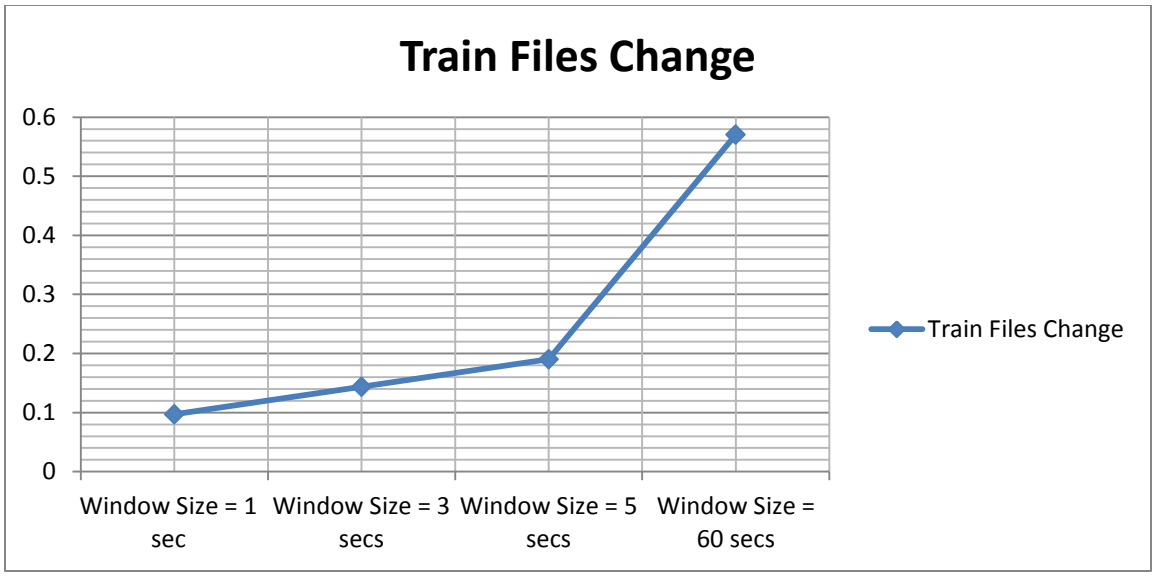


Figure 60 Percentage change for 10 users to train files resulting with window algorithm

Summary of Results

A total of 614 experiments were conducted for this study as reported below:

- 330 experiments using synthetic data without context drift
- 132 experiments using synthetic data simulating context drift
- 42 experiments using network data from a real cellular network
- 42 experiments simulating DOS attacks
- 42 experiments simulating Phish attacks
- 18 experiments simulating different session identification approaches
- 8 experiments covering user accuracy scalability

A consistent result across all experiments conducted in this study is the fact that HTM algorithms always outperform in terms of accuracy alternate algorithms (1st and 3rd Order Markov chains, All-K, PPM). More specifically, for experiments conducted with synthetic data

which used no context drift, the difference in accuracy performance between HTM algorithms and alternate algorithms is substantial. Alternate algorithms with one day worth of test data never produced recall statistics above 42% (see Figure 32,Figure 33, Figure 34) and when the test data set consisted of 3 observations, these algorithms never produced recall statistics over 66%. In contrast, HTM algorithms produced accuracy statistics (recall statistics) as high as 99% for a sample of 100 users as shown in Figure 33.

The accuracy of HTM algorithms improves as the number of web destinations visited increases beyond 1000 to 5000 and 10,000. Recall results for experiments conducted with synthetic data without introducing simulated context drift, produced recall values of 99% for 500 users as shown in Figure 37 and Figure 39.

Overall the best performing HTM algorithms in terms of accuracy and scalability are the Bottom Up for HTM layers 1 and 3 and the TopTop algorithms (see section "Accuracy Scalability" for more details). For alternate algorithms the 3rd Markov chain always tends to perform more poorly than the other algorithms.

Experiments conducted with synthetic data which simulated concept drift show that the HTM accuracy is mostly impacted by concept drift in test data (reducing accuracy by as much as 25%) that tends to split sequences of destinations often as shown when the "random walk" algorithm is utilized to apply concept drift. On the other hand concepts drift that tends to preserve a portion of the original sequence of destinations but adds new connections to existing destinations or new destinations to existing connections, has much smaller of an impact on the HTM accuracy (reducing accuracy by no more than 11%) .

Several experiments were conducted with real network data collected from a CDMA cellular operator's network. The data collected showed very large number of repeated destinations within observations making it difficult for the HTM to infer unique patterns. The intra observation repetitiveness reported for some users is as high as 94% (see Figure 47) meaning that on average only 3 web destinations in 50 are unique within an observation. Accuracy as reported by the HTM1 (version 1) was as low as 32% and the highest accuracy reached was 81%, as shown in Figure 44, but most HTM algorithms performed poorly.

A new version of the HTM, (HTM2) was created which accounted for these repeated patterns. In addition, the real data set used for experiments was modified to remove multiple repeated destinations that occurred with the exact same timestamp. Results improved for all HTM algorithms, by raising the worst accuracy recorded to 61% as shown in Figure 52 and reaching recall values as high as 95% as shown in Figure 51. However, the HTM is still underperforming with respect to the accuracy performance measured against the synthetic baseline.

Experiments which simulated DOS and Phish attacks were also conducted. These experiments showed that the HTM algorithms are minimally impacted by these attacks. Most HTM algorithms keep accuracy from decreasing by more than 11% after a DOS attack and 5% after a Phish attack.

The last set of experiments conducted in this study considered the impact of using different session identification algorithms to train the HTM. These experiments showed that the sliding window session identification algorithm 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. To put things in perspective, a window size of 1 minute, produces a change in the test file (loss of web destinations) from the original train file of 57% yet recall is reported at 92%, compared to the best performing TCP timestamp (with 10% data loss and 10% device resets) which reports a recall value of 83% as shown in Figure 59.

The results reported by these experiments clearly support the goals of this study by showing that a hierarchical temporal memory produces better accuracy results than alternative traditional Markov based approaches. HTM results consistently outperformed alternate algorithms regardless of the algorithm chosen and the data set used (synthetic or real network data). Accuracy scalability, that is the ability of the solution to maintain accuracy as number of users/destinations increases, was also shown to be superior for the HTM over alternative Markov based approaches (see section "Results of experiments to verify user attribution accuracy without concept drift using synthetic data" for more details).

Chapter 5

Conclusions, Implications, Recommendations, and Summary

This study has set out to address the user attribution problem which attempts to identify communication traffic that belongs to a user, as the user possibly moves across networks, when the information needed to identify those users is missing. The experiments conducted in this study have shown that the hierarchical temporal memory (HTM) is quite accurate in correctly identifying time based patterns that represent user navigational patterns. Test results have shown that the HTM can provide reliable user attribution in scenarios where malicious users attempt to access network resources by performing simulated Phish and DOS attacks. The effects and impacts of mobility so critical in the user attribution problem, was tested against different session identification algorithms.

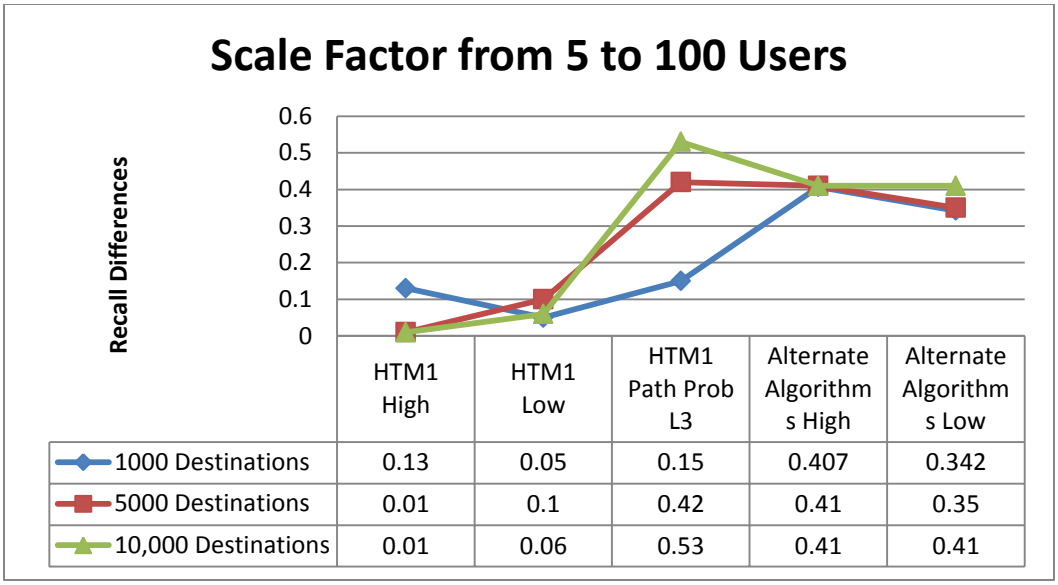
The results of the experiments conducted for this study are promising. A recurring theme in this study shows that alternate algorithms based on the traditional implementation of Markov chains consistently underperformed HTM algorithms pretty much in all experiments. The experiments conducted in this study bring to bear very good accuracy and accuracy scalability results with synthetic data and good results with real network data thus satisfying the original goals of this research. There is a strong belief that even better accuracy results can be achieved when processing real network data with specialized algorithms built within the HTM designed to address specifically extremely repetitive patterns in observations. Experiments also show that the HTM tolerates quite well noise in test data in the form of either concept drift or DOS and Phish attacks. The lack of real network TCP timestamps limited session identification experiments to utilizing synthetically created TCP timestamps however the experiment did show the merits of

the sliding window algorithm as an accurate session identification algorithm. It would be a topic of further study to determine how well the sliding window, source IP address and TCP timestamp algorithms correctly identify sessions with real network data. The experiments also brought to bear the fact that the HTM run-time performance scalability, as the number of users and the amount of test data increase, is not good and represents an area of further investigation.

In this study synthetic data as produced by the input generator algorithm described in section "Generation of Synthetic Data for the Simulation" represented a reasonable data set to run experiments since similarity statistics between train and test data sets showed that the test data generated for a given user is relevant to the train data and yet independent enough to simulate realistic experiments.

All experiments run with synthetic or real data, which compare HTM algorithms and alternate algorithms (implemented based on traditional Markov chains), show the HTM outperforming alternate algorithms as was described in Chapter 4. It is important to note that all 7 HTM algorithms implemented for this study share the same degree of membership calculation of longest common subsequence, longest common substring and sequence persistence to determine the similarity of input against learned sequences. These HTM algorithms differ in how they combine and process the results of multiple degrees of membership calculations within and across HTM layers for multiple observations.

Accuracy reported by HTM algorithms also scales better with increasing number of users and web destinations than the accuracy reported by alternate algorithms as shown in Figure 61.



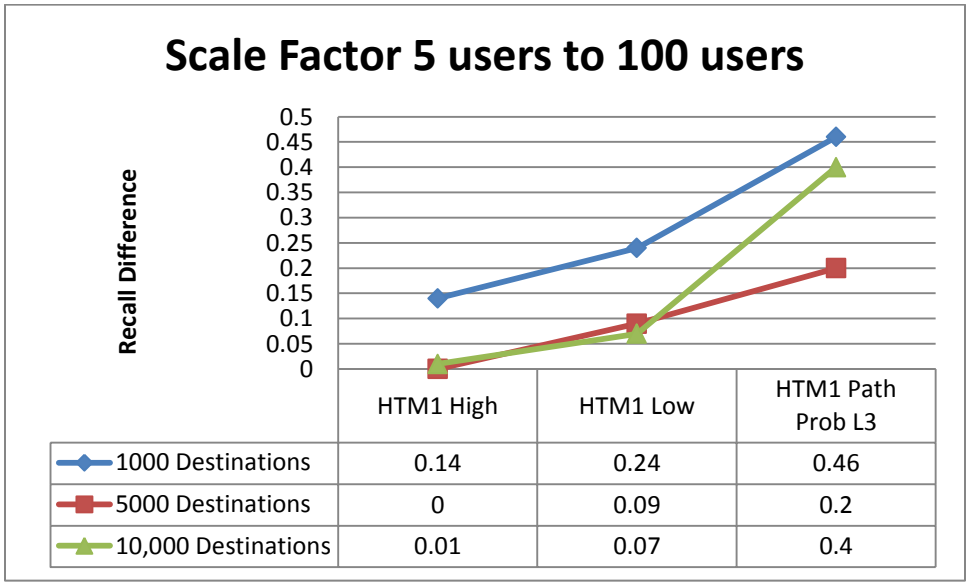
	HTM1 High	HTM1 Low	HTM1 Path Prob L3	Alternate Algorithms High	Alternate Algorithms Low
1000 Destinations	0.13	0.05	0.15	0.407	0.342
5000 Destinations	0.01	0.1	0.42	0.41	0.35
10,000 Destinations	0.01	0.06	0.53	0.41	0.41

Figure 61 Recall Accuracy Scaling for up to 100 users for 1 test day

Figure 61 shows the difference in recall accuracy between experiments run for 5 to 100 users over 5 train days and 1 test day using synthetic data. The high and low recall values of all HTM algorithms (excluding path probability for layer 3 shown as an outlier) and alternate algorithms was recorded and the difference between values for 5 and 100 users was tabulated in Figure 61. It can be seen that excluding the layer 3 path probability algorithm, HTM algorithms scale better than alternate algorithms with a maximum of 13% loss in accuracy when tracking 1000 web destinations and moving from 5 to 100 users compared to 41% loss in accuracy for alternate algorithms. For 5000 and 10,000 web destinations the scale factor for the HTM algorithm is as low as 1% for high recall values.

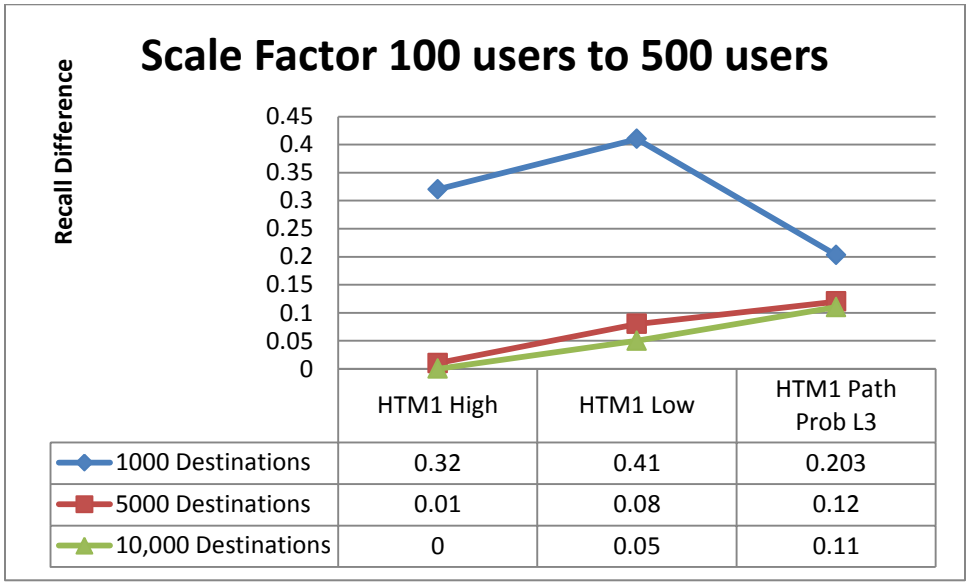
The HTM scales well also in experiments where the number of users grows to 500 using 5 days of synthetic train data and 3 observations for test data. Figure 62 and Figure 63 show that the recall difference between high and low recall values moving from 5 users to 100 users and then to 500 users for 5000 and 10,000 web destinations is about the same and stays below 10%.

For high recall values the scale factor for 5000 and 10,00 web destinations shows perfect scaling with a scale factor value as low as 0%. Results of experiments run with synthetic data show high levels of accuracy for ranges of web destinations above 5000. This range of destinations visited by users matches the range found in the real network data set used in the experiments of 5200 web destinations.



	HTM1 High	HTM1 Low	HTM1 Path Prob L3
1000 Destinations	0.14	0.24	0.46
5000 Destinations	0	0.09	0.2
10,000 Destinations	0.01	0.07	0.4

Figure 62 Recall Accuracy Scaling for up to 100 users with 3 observations



	HTM1 High	HTM1 Low	HTM1 Path Prob L3
1000 Destinations	0.32	0.41	0.203
5000 Destinations	0.01	0.08	0.12
10,000 Destinations	0	0.05	0.11

Figure 63 Recall Accuracy Scaling for up to 500 users with 3 observations

More targeted experiments were also conducted with synthetic data to measure accuracy scalability specifically from a user point of view using a fixed optimal number of web destinations (5000) and leveraging the two best HTM algorithms (BottomUp layer 1 TopTop layer 3). For these experiments the number of users was: 150, 250, 350 and 450. Results show that increases in the percentage of the number of users from 11% (from 450 to 500) to 67% (from 150 to 250) result in accuracy (for both recall and precision) percentage change values that stay within the very small range of 1.5% to -0.5% while producing consistently high accuracy values in the range of 95% to 99% (see section "Accuracy Scalability" for more details).

One of the biggest challenges faced when running experiments with 500 users was the need to reduce the experiment run times. 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 Graph 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 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, 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 the run times reported below in Figure 64 and Figure 65. It can be seen that as the

number of users in conjunction with the test input size increase the run-times do increase dramatically.

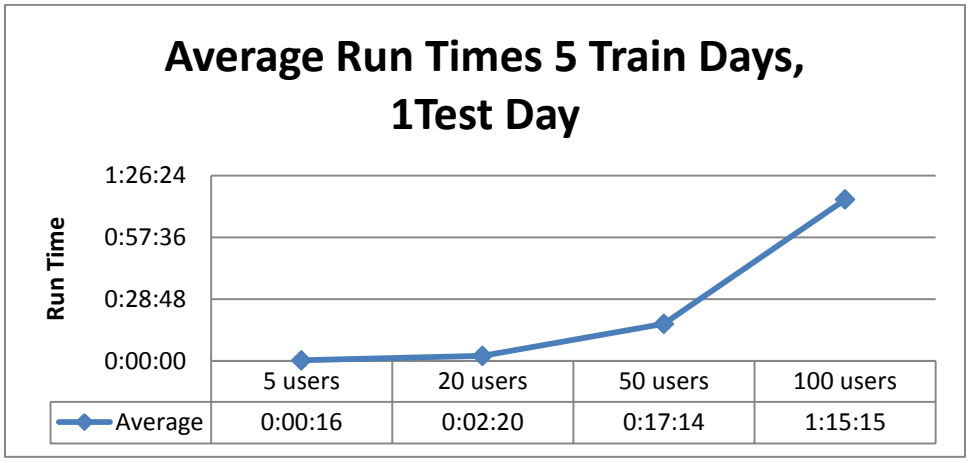


Figure 64 HTM Run-Times for 1 Test Day

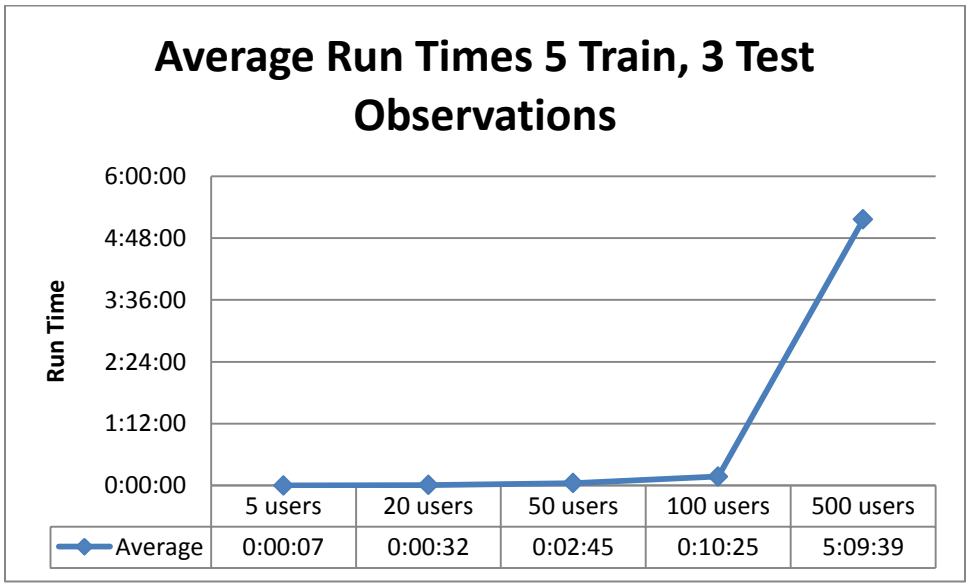
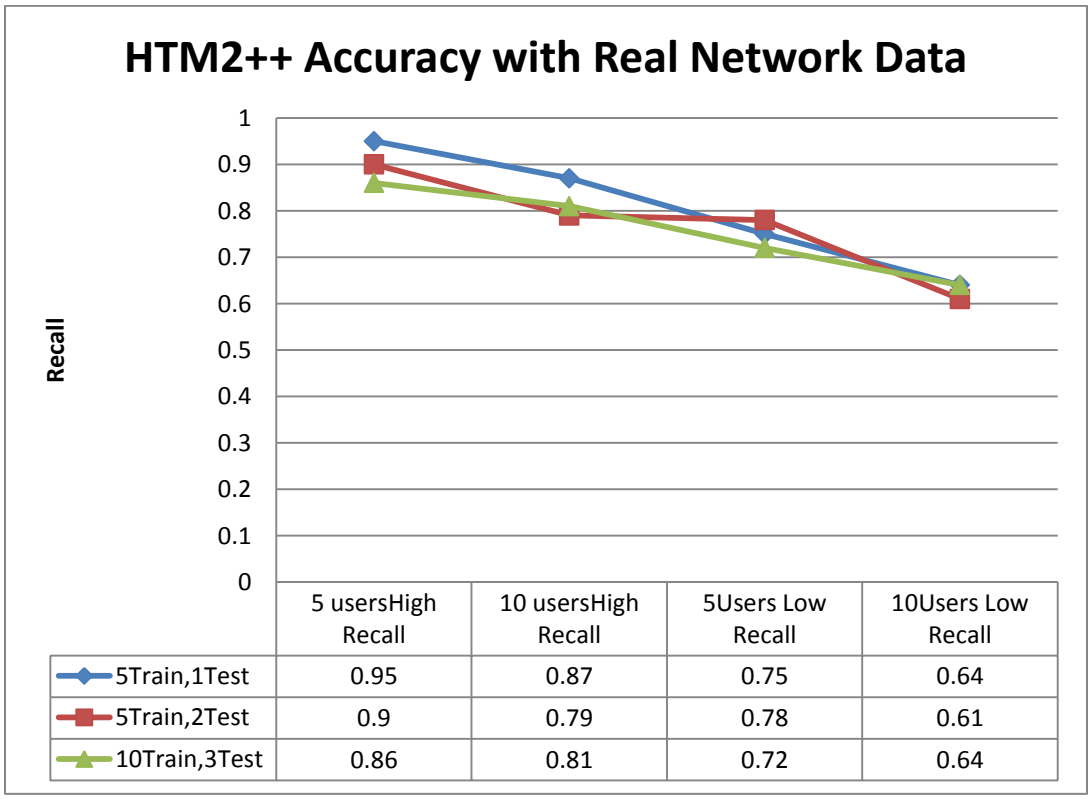


Figure 65 HTM Run-Times with 3 observations

The amount of RAM main memory used by the HTM 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 Figure 8) to receive 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 coming to a near halt.

The performance recall accuracy of the HTM as reported by experiments with real network data is good as shown in Figure 66 which reports high and low recall values for these experiments. However, performance still falls short of what has been reported for similar experiments with synthetic data where observations do not show extreme levels of repeated destinations.



	5 usersHigh Recall	10 usersHigh Recall	5Users Low Recall	10Users Low Recall
5Train,1Test	0.95	0.87	0.75	0.64
5Train,2Test	0.9	0.79	0.78	0.61
10Train,3Test	0.86	0.81	0.72	0.64

Figure 66 HTM2++ Recall Accuracy with real network data

Further study is needed to explore new HTM algorithms that specifically address the high levels of repeated destinations found in real network data. In text mining, a similar problem exists with commonly occurring words like “the”, which impact negatively the accuracy of text inference algorithms. To address this problem *inverse document frequency* is utilized which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. As an area of further research, a new HTM algorithm could be developed that could leverage a similar concept.

Experiments also show that the HTM handles concept drift well in test data which modifies the last portion of a sequence of learned destinations to the point where often it tends to increase accuracy as shown in Figure 67.

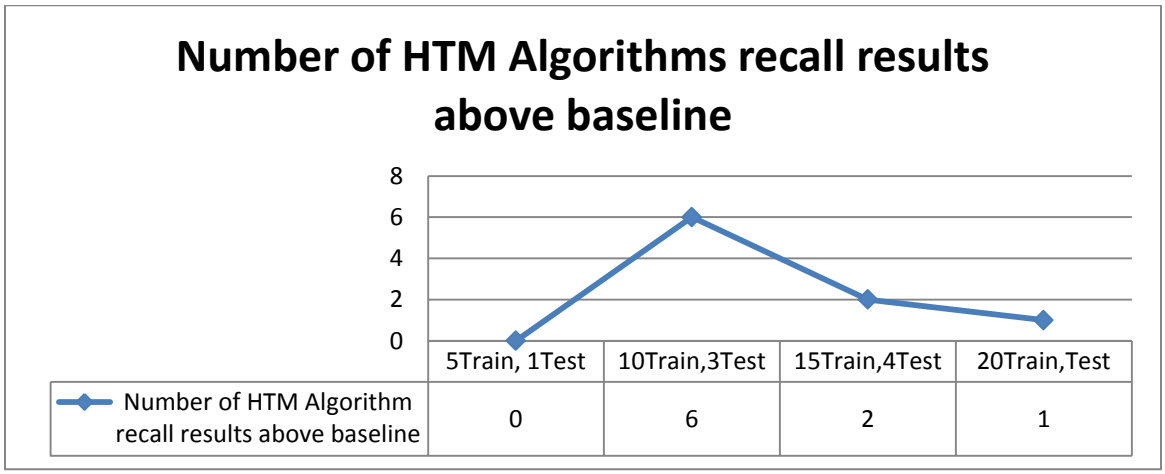
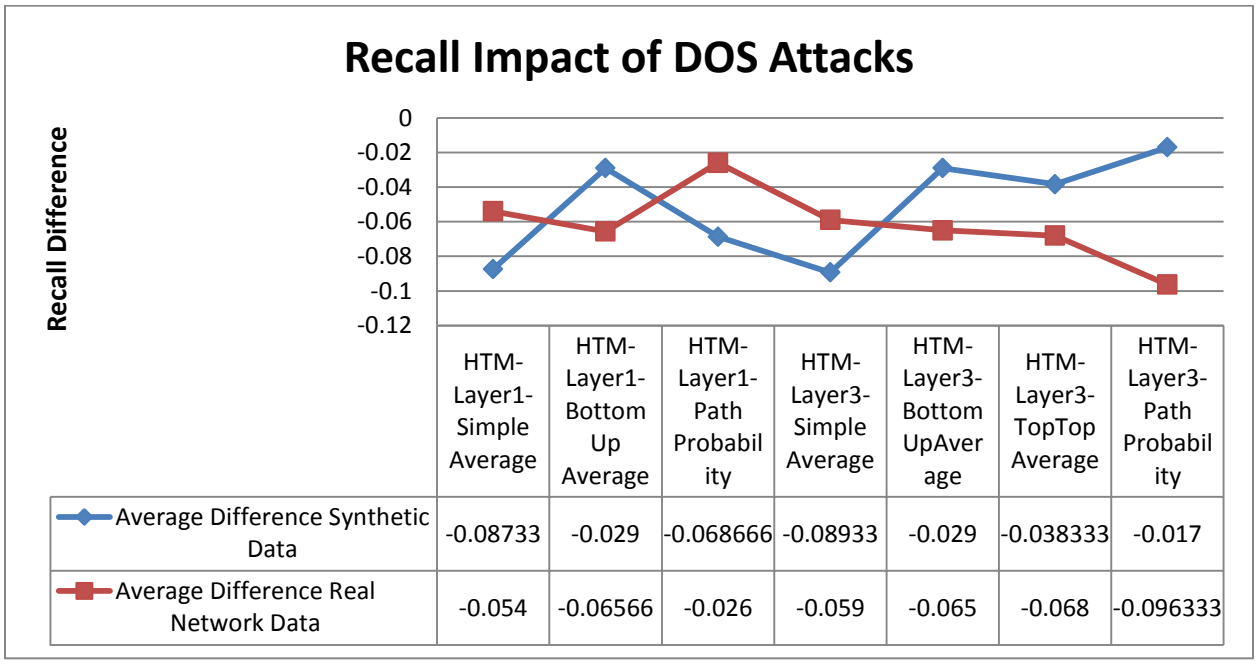


Figure 67 HTM Recall Accuracy in the presence of concept drift above base line

After applying concept drift to test data in the form of 20% new connections to existing destinations and 10% new connections to not learned before destinations, the figure above shows that in 3 out of 4 sets of experiments the HTM accuracy of several algorithm actually improves

over the baseline which used no concept drift. Because this type of concept drift tends to change the last portion of an existing learned sequence it is possible that an existing sequence in the train data set is made even more unique by the changes affected by this form of concept drift. In contrast, concept drift that tends to split sequences of learned destinations, as done by the random walk algorithm, has a consistently negative impact on HTM accuracy performance with recall loss of up to 25% compared to the baseline (see Figure 43).

The accuracy performance of the HTM does also relatively well in experiments which simulate DOS attacks with synthetic and real data as shown in the figure below never exceeding a 10% recall impact recorded after the attack.



	HTM-Layer1-Simple Average	HTM-Layer1-Bottom Up Average	HTM-Layer1-Path Probability	HTM-Layer3-Simple Average	HTM-Layer3-Bottom UpAverage	HTM-Layer3-TopTop Average	HTM-Layer3-Path Probability
Average Difference Synthetic Data	-0.08733	-0.029	-0.068666	-0.08933	-0.029	-0.038333	-0.017
Average Difference Real Network Data	-0.054	-0.06566	-0.026	-0.059	-0.065	-0.068	-0.096333

Figure 68 Recall Accuracy Impact of a DOS Attack

Experiments which simulated Phish attacks against the HTM show even less of an impact than for DOS attacks, except for layer 3 path probability, after the attack as shown in Figure 69.

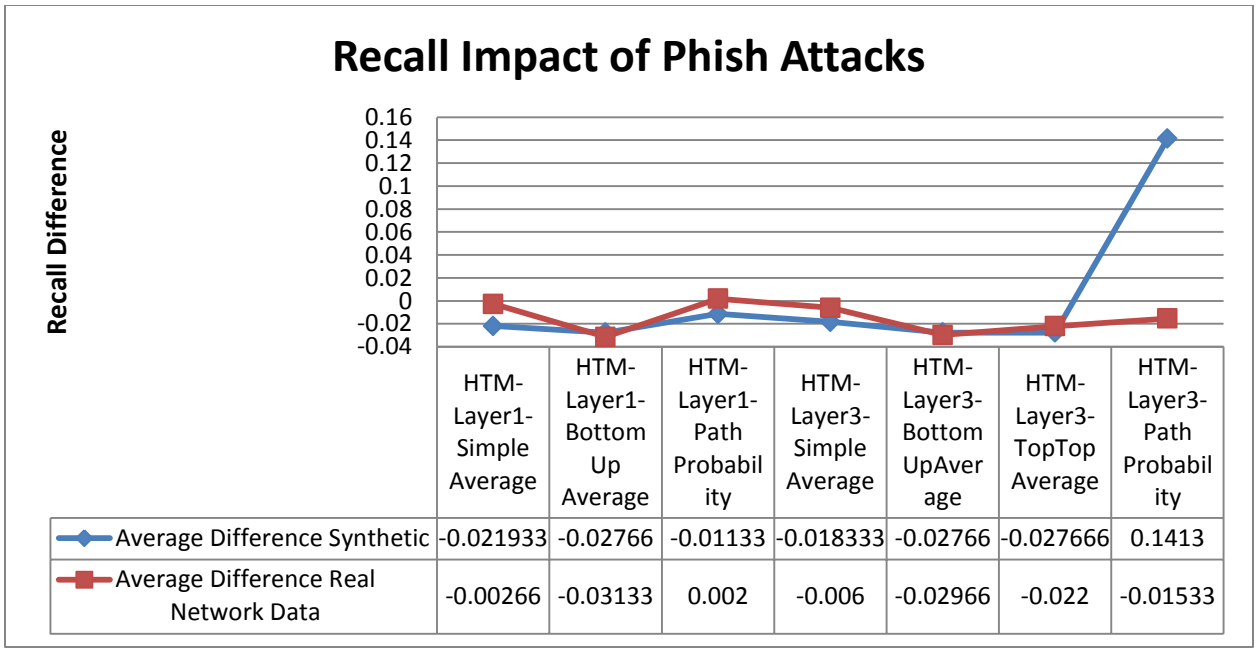
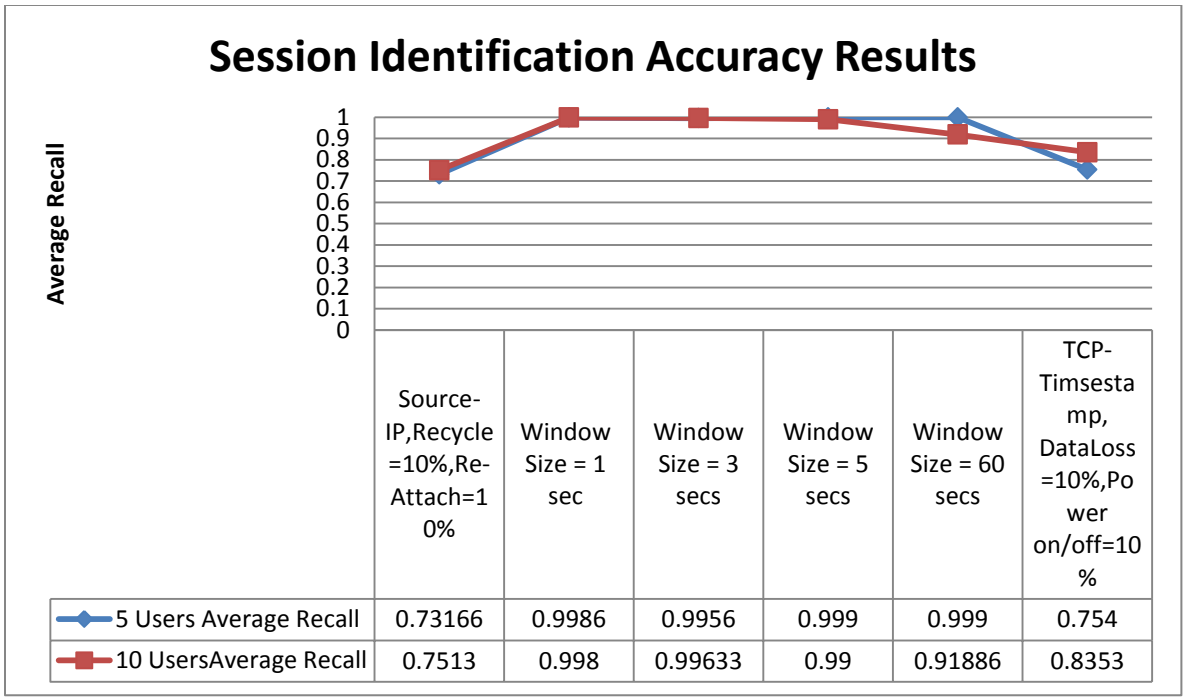


Figure 69 Recall Accuracy Impact of Phish Attacks

Results from the session identification tests show that the sliding window approach used without simulated noise conditions is a more accurate session identification algorithm when used to train the HTM than the TCP Timestamp and source IP address algorithms used under simulated noise conditions as shown in Figure 70.



	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 70 Session Identification Recall Accuracy Results

These session identification results show the dominance of the sliding windows algorithm which as opposed to other session identification algorithms was not subjected to noise conditions. While real network data was used for these experiments to provide realistic timing for the sliding window algorithm, it would be an important extension of this study to utilize real network data which contains changing IP addresses and real TCP timestamps, which were missing in these experiments, and run these session identification algorithms against the sliding window algorithm.

The are several contributions that this study has made in extending the hierarchical temporal memory model originally proposed by George and Widrow (2008) which was not designed to support sequences. All of the following extensions represent contributions to the field and were designed to improve HTM inference accuracy.

- This study implemented 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 a variety of HTM inference algorithms.
- This study identified the limitations of traditional state cloning and proposed "sequence cloning" as a technique to address its shortcomings and improve inference accuracy
- This study introduces the concept of "playback" to distribute accurately learned sequences from lower to higher layers of the HTM to reduce learning times and improve inference accuracy

Another area of this study that deserves a more in depth analysis is the way the HTM splits sequences during both learning and inference phases. Experiments that introduce concept drift using the random walk algorithm have shown that the HTM accuracy degrades substantially when sequences of web destinations learned from the train data set are split in the test data set. The current implementation of the HTM terminates a sequence and starts another under the following conditions:

1. A fixed maximum input size has been processed
2. A maximum *learned* inter destinations arrival rate is exceeded
3. The same destination is already present in the sequence (HTM version 1). HTM version 2 also uses condition (3) to split a sequence but continues to process repeated destinations already in the sequence until a new destination not already in the sequence is encountered or one of conditions (1) or (2) are met.

Manipulation of these conditions or the parameters used by these conditions impacts HTM accuracy performance as was shown for version 2 of the HTM. In addition, preliminary experiments conducted by disabling and enabling learning for inter destination arrival time showed improvements (e.g. 5% accuracy improvement for the TopTop HTM algorithm when inter destination arrival rate learning is enabled versus disabling it and relying on a fixed inter destination arrival time). Learning for inter destination arrival time was implemented using a K means clustering algorithm to learn inter destination arrival rates.

A limitation of this study is the restricted number of users (10) that was utilized with real network data experiments. Fifty users were tracked over a period of one month, but due to of large periods of traffic inactivity only 14 users provided enough data to support realistic experiments. It would be beneficial to run experiments with larger number of users as part of future research that addresses the high levels of repeated destinations found in real network data.

Known Limitations of Proposed Approach

Each time a device is recycled (powered on and then powered off), that device looks like a new device entering the network even if the device has been previously recognized by the prototype. This is because the TCP time stamp value of a given device is always reset to either a given fixed value or a random value and thus it will not match the TCP time stamp value associated with existing HTMs. These HTMs have their TCP time stamps continuously updated with the passage of time from the time the user of the device first powered on the device and entered the network. A device that having being powered on and having entered the network, powers off and then powers back on and reenters the network will cause the session identification algorithm to mistake this as a new user session (not seen before) and will create

duplicate instances of an HTM for the same user. This occurs only if a user recycles his/her device after having entered the network. This problem is similar to the situation where the same user makes use of two different devices to access the network. Even if the user never powers off both devices, at least 2 HTM instances are created for the same user. A solution to address this limitation is proposed below as research to be conducted in a further study.

When devices get recycled (powered on and off and then on again) the source identification algorithm used during training mistakes a known device (source) for a new one and will mistakenly create multiple instances of HTMs for the same user. A solution to address this problem, shown in Figure 71, leverages the idea that HTMs representing the same user, when presented with the same input, are likely to generate similar inference at layer 3 which helps identify duplicate HTMs representing the same user.

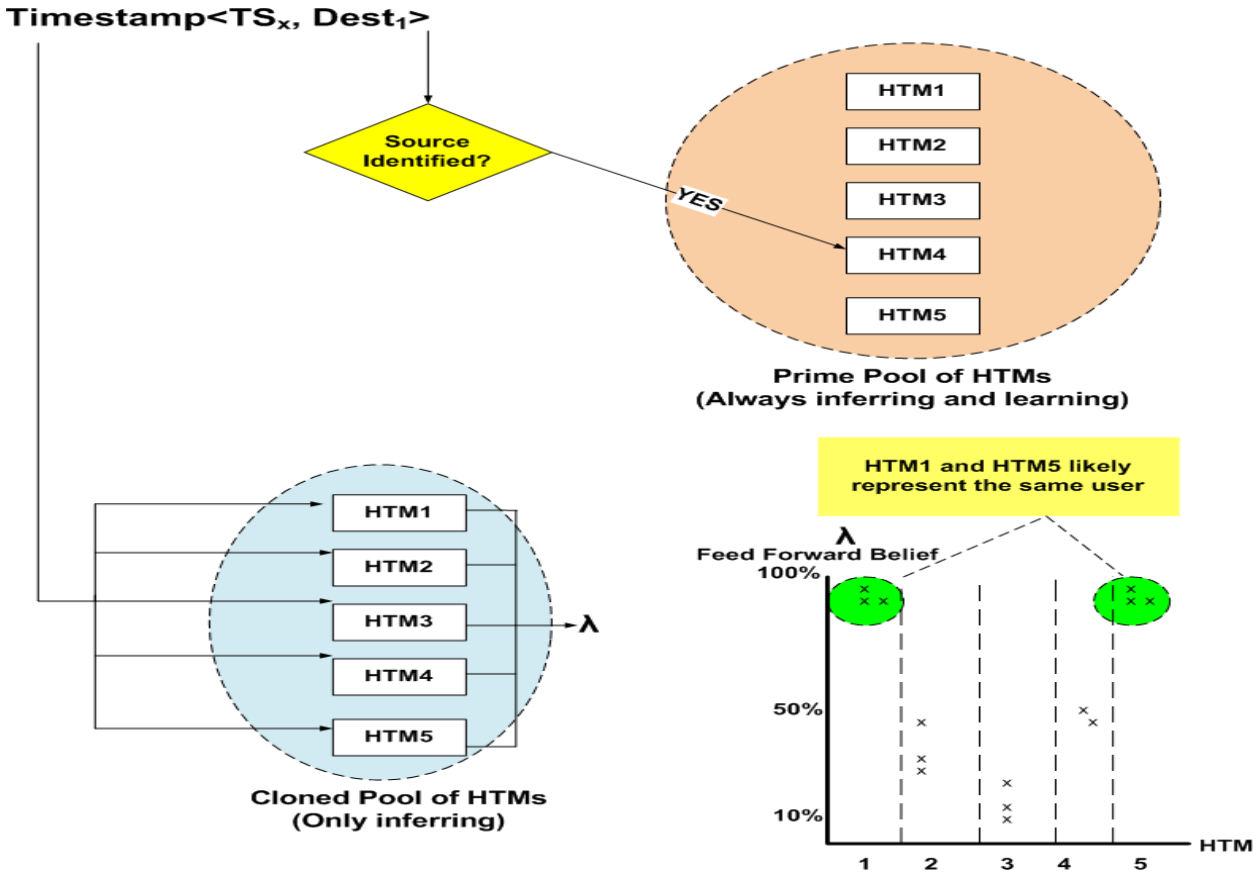


Figure 71 Detecting Duplicate HTMs

The high level algorithm below depicts how duplicate HTMs are detected. This algorithm removes HTMs likely to belong to the same user and keeps the oldest HTM which is likely to belong to the original user and have his/her longest and most representative behavioral history.

WHEN a given number of HTMs in the Prime HTM Pool have reached the "inference" stage since the last HTM cloning procedure THEN

- Start the HTM Cloning Procedure:
  - Flush the HTM Clone Pool
  - Clone all HTMs in "inference" state from the Prime HTM pool
  - Disable learning for all of the HTMs in the HTM Clone Pool so that these HTMs only operate in inference mode

-Feed any single input received by the prototype to all HTMs in the HTM Clone Pool
-Collect the output (feed forward beliefs) of all HTMs from the all HTMs in the HTM Clone Pool into clusters based on similarity of feed forward beliefs outputs
-For Each cluster select *k* clone HTMs with the most similar feed forward beliefs readings
- Of the *k* clone HTMs select all of k-1 HTMs but the oldest cloned HTM
- Delete from the Prime HTM Pool the equivalent (twins/clones) HTMs identified From the cloned HTM Pool in the previous step

Figure 72 Algorithm to detect duplicate HTMs

Summary

The problem of tracking the behavior of users without explicit identifiers is very relevant and "challenging" because many ISPs assign customers dynamic IP addresses that change periodically as reported by Hermann, Banse and Federrath (2013). The authors acknowledge that user behavior tracking is feasible. Their experiments show accuracy results with up to 85% recall for large number of users (over 3000). However, the authors concede that recall accuracy degrades when the source IP address changes frequently. For instance, when the source IP address changes every 3 hours recall accuracy drops to 65%, when it changes each hour recall accuracy drops to 54% and when it changes each 30 minutes recall accuracy drops to 42%.

This study has addressed the same basic problem as proposed by Herrmann et al. but from the perspective of user attribution in the context of user mobility across complex networks. It is when users move within and across networks that the problem described by Herrmann et al. becomes more difficult to tackle since reliance on explicit identifiers such as source IP address become ineffective as the source IP address changes periodically within mobile networks (as is the case for cellular networks) and across networks, each time a user attaches to a new network (e.g. WIFI hotspots or cellular network).

This study confirmed with synthetic and real network data that past user communication behavior can be used as a predictor of future user communication behavior even when user behavior changes over time due to natural concept drift. This study confirmed the power law distribution of real network data with few web sites being visited often. This research also confirmed that the presence of long tail web sites (rarely visited) among many repeated destinations can create unique differentiation. Synthetic data generated using a modified version of Price's model (1976) for networks creation enabled generation of test data that was relevant to a corresponding train data set and independent enough to support realistic experiments. What was not anticipated prior to the experiments was the high degree of repetitiveness of some web destinations found in real network data.

The experiments conducted in this study have shown that a hierarchical temporal memory (HTM) which learns and infers sequences of web destinations leveraging multiple layers (to learn and infer even longer sequences) has proven to be an effective framework for developing user attribution inference algorithms. Experiments have shown that the HTM can provide high levels of accuracy using synthetic data with 99% recall accuracy for 100 and 500 users and good levels of recall accuracy of 95 % and 87% for 5 users and 10 users respectively when using real

network data. Experiments results show that HTM weighted average algorithms in the form of Bottom-Up and TopTop tend to outperform all other HTM algorithms for both synthetic and real network data. In addition, the fact that TopTop is an HTM layer-3 only algorithm brings to bear the improved accuracy that can be achieved when using multiple HTM layers. While accuracy results were positive for most experiments, run-time performance with increasing test data set sizes beyond 150 destinations for more than 500 users proved to be poor and represent an area of future research.

Experiment results consistently showed that HTM algorithms outperformed alternate traditional Markov chain based algorithms in all experiments. However, when running calibration tests for real network data alternate algorithms outperformed HTM based algorithms as shown in Table 17. A possible reason for this result can be attributed to the fact that all alternate algorithms perform exact or partial "matches" of the context preceding the current input (sequence of web destinations). A partial match is based on exact matching of a shorter substring of the original context. HTM algorithms instead seek the best longest common subsequence within learned web destinations and leverage the concept of "similarity" where the input need not match exactly the context or be an exact substring of it, instead the input needs to just contain some of the same destinations in the same order as was previously learned by the HTM.

Experiments have also shown that the HTM does not need much data (as little as 150 web destinations) to accurately identify users even when the number of user is as high as 500. This can have important implications in the area of network communication security where malicious users need to be quickly identified in order to be stopped using as little data as possible. It is important to note that using identifiers like cookies and source IP addresses to solve the user attribution problem can expose privacy concerns. This occurs when these identifiers are used to

discover the real identity of the user who just logged into the web site that assigned the cookie to the user or to discover the user who was authorized access to the cellular network after being assigned a specific source IP address. Because the HTM forgoes use of identifiers to address the user attribution problem it can recognize users over time without revealing their true identity and thus be able to maintain high levels of privacy.

The HTM showed to be fairly resistant to noise in the form of concept drift, denial of service (DOS) and Phish attacks. Specifically, experiments conducted with synthetic data which simulated concept drift show that the HTM accuracy is mostly impacted by concept drift in test data (reducing accuracy by as much as 25%) that tends to split sequences of destinations often as shown when the "random walk" algorithm is utilized to apply concept drift. On the other hand concepts drift that tends to preserve a portion of the original sequence of destinations but adds new connections to existing destinations or to new destinations, has much smaller of an impact on the HTM accuracy (reducing accuracy by no more than 11%) . Experiments which simulated DOS and Phish attacks were also conducted. These experiments showed that the HTM algorithms are minimally impacted by these attacks. Most HTM algorithms keep accuracy from decreasing by more than 11% after a DOS attack and 5% after a Phish attack. These results show promise for possible utilization of the HTM in a network security environment as part of an intrusion detection and prevention solution.

How credible are these results? The HTM prototype (version V1 and V2) and the alternate approaches are completely written from scratch in Java. What was done to minimize the risk of introducing errors into the logic of the model and code which could bias the results of the experiments conducted in this study?

1. Extensive self-verification code was implemented within the HTM (e.g. code to verify the integrity of Markov graphs and Markov chains at each layer of the HTM) as outlined in section "Validating the Instrument"
2. "Calibration" which verifies that the HTM and alternate algorithms can always recognize the input they were trained with. Each version of the HTM (versions V1 and V2) and alternate algorithms were qualified against calibration tests before being run against real test scenarios as shown in Appendix H. A prerequisite for running the HTM against different experiments was to achieve 100% recall accuracy for experiments that used synthetic network data for all HTM algorithms both at layers 1 and 3 and for alternate algorithms. For real network data it was not possible to achieve 100% recall accuracy due to the high levels of repetitiveness of the input, as a result using HTM version V2 qualification was established with recall values as low as 98% for HTM algorithms at layer 1 and 93% for HTM algorithms at layer 3 (see Appendix H for more details).

The ability to learn and infer when using streaming network data has been an objective of this study. To this end the HTM leveraged unsupervised learning by utilizing TCP timestamps embedded in the input stream. Due to the lack of TCP timestamps in real network data traces, synthetic timestamps were successfully utilized for this research. Further experiments utilizing different session identification algorithms run against synthetic data were also performed with a new sliding window algorithm showing promising results. Session identification experiments would benefit from further study which would utilize real network data especially due to the critical nature that session identification plays in the user attribution problem as reported by several authors in the literature. Yang (2010) acknowledges that her results cannot scale to large

number of users due to the inability of her session identification algorithm to link up multiple sessions belonging to the same user. Herrmann et al. (2013) also reports substantial decrease in accuracy when the source IP address used to identify users in a session changes often.

In order to appreciate the relevance of this work one needs to consider that the internet of people is becoming the internet of things where mobility is one of the driving forces. METIS[1], Mobile and wireless communications Enablers for the Twenty-twenty (2020) Information Society is a large EU co-funded research project created in 2012. The project objective was to respond to societal challenges for the year 2020 and beyond by laying the foundation for the next generation of the mobile and wireless communications system. METIS is a consortium of 29 partners spanning telecommunications manufacturers, network operators, the automotive industry and academia. METIS has defined 5G networks of the future as possessing the following key features:

- Massive machine communication
- Moving networks not just moving users and moving devices
- Ultra dense networks which utilize a variety of access technologies (e.g. WIFI, Cellular, Bluetooth, etc.)

A key take away from this view is that mobility will 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. This research represents an encouraging first step towards addressing the user attribution problem in a mobile environment that covers multiple complex networks.

[1] This definition comes from METIS Fact sheet available at www.metis2020.com

Appendix A

HTM1 User Attribution Test Results Using Synthetic Data with no Concept Drift

Five train days, one test day and one thousand destinations

The next graph supports precision statistics for 5 train days, 1 test day and 1000 destinations.

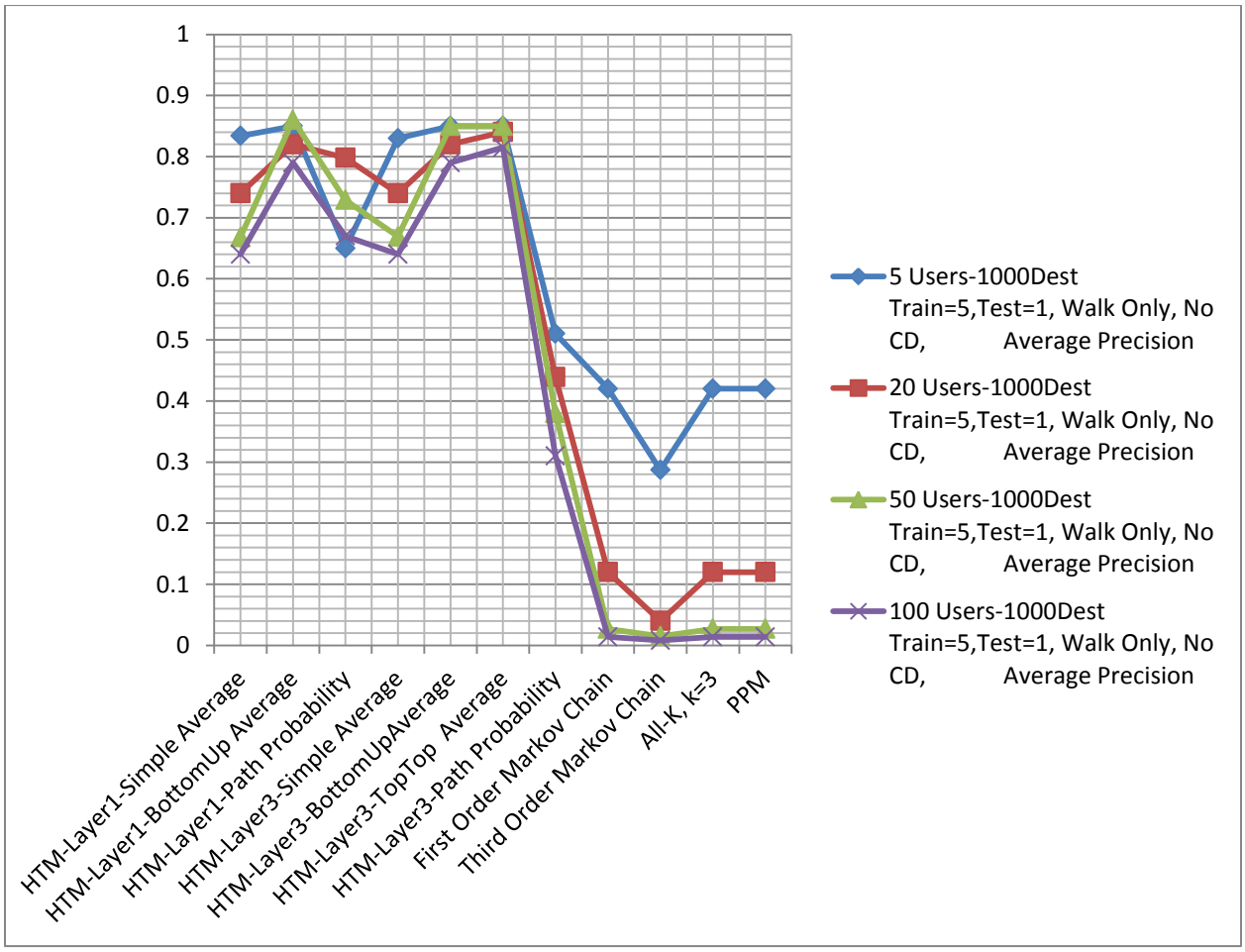


Figure 73 Appendix A Synthetic Data Precision Results for 5 Train, 1 Test, 1000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 1 test day and 1000 destinations.

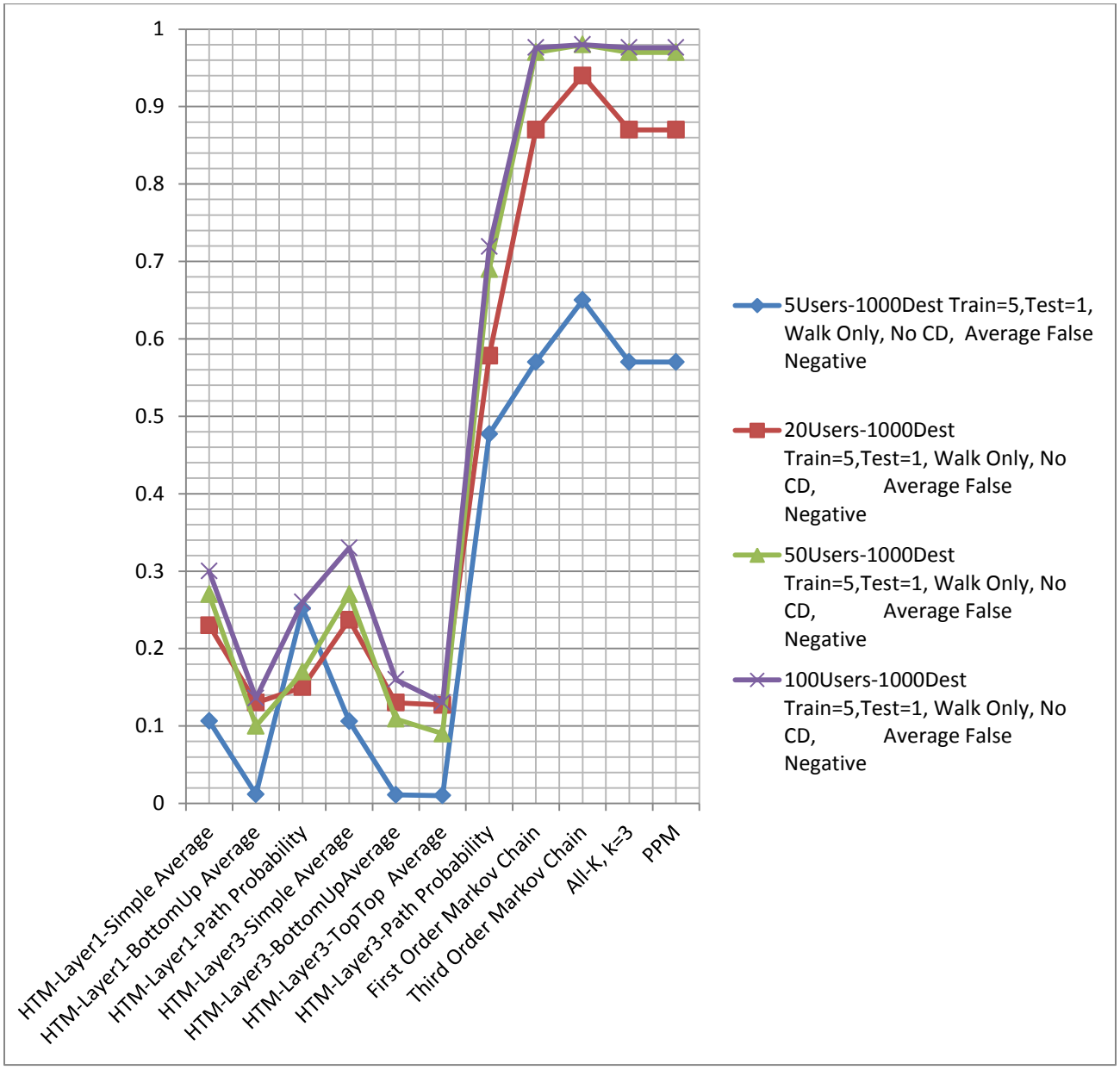


Figure 74 Appendix A Synthetic Data False Negatives Results for 5 Train, 1 Test, 1000 Destinations

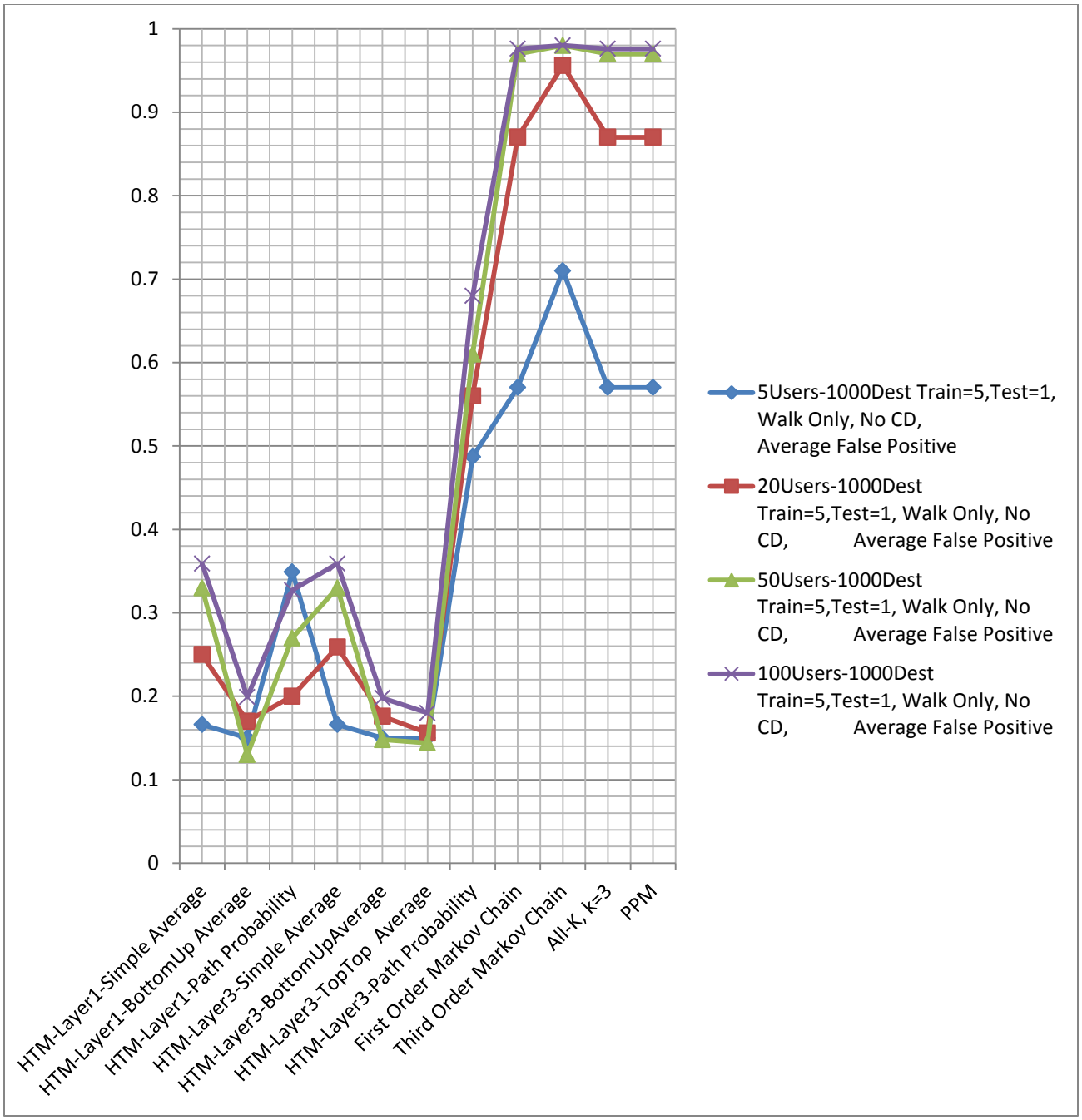


Figure 75 Appendix A Synthetic Data False Positives Results for 5 Train, 1 Test, 1000 Destinations

Five train days, one test day and five thousand destinations

The graph below supports the precision statistics for 5 train days, 1 test day and 5000 destinations.

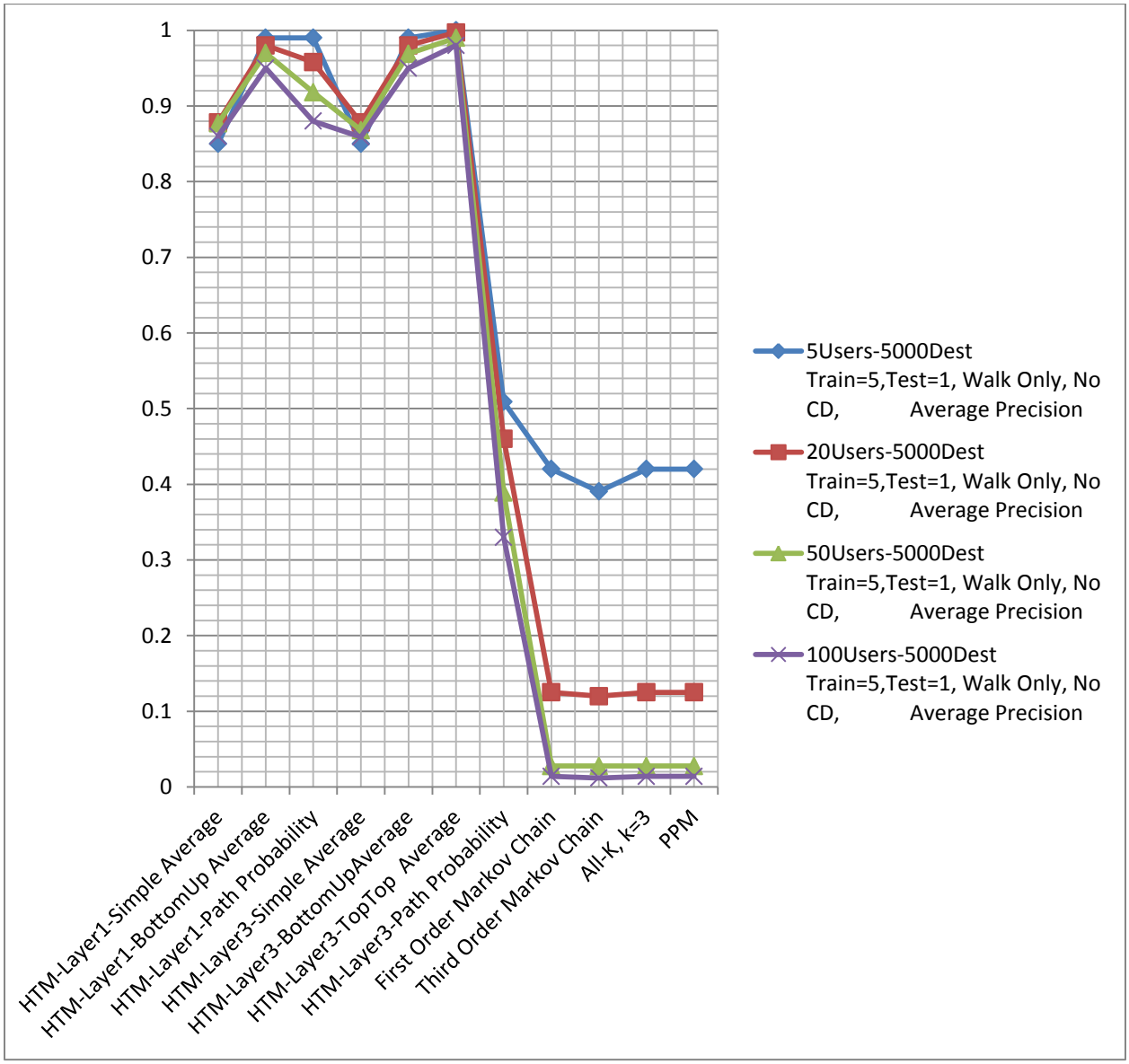


Figure 76 Appendix A Synthetic Data Precision Results for 5 Train, 1 Test, 5000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 1 test day and 5000 destinations.

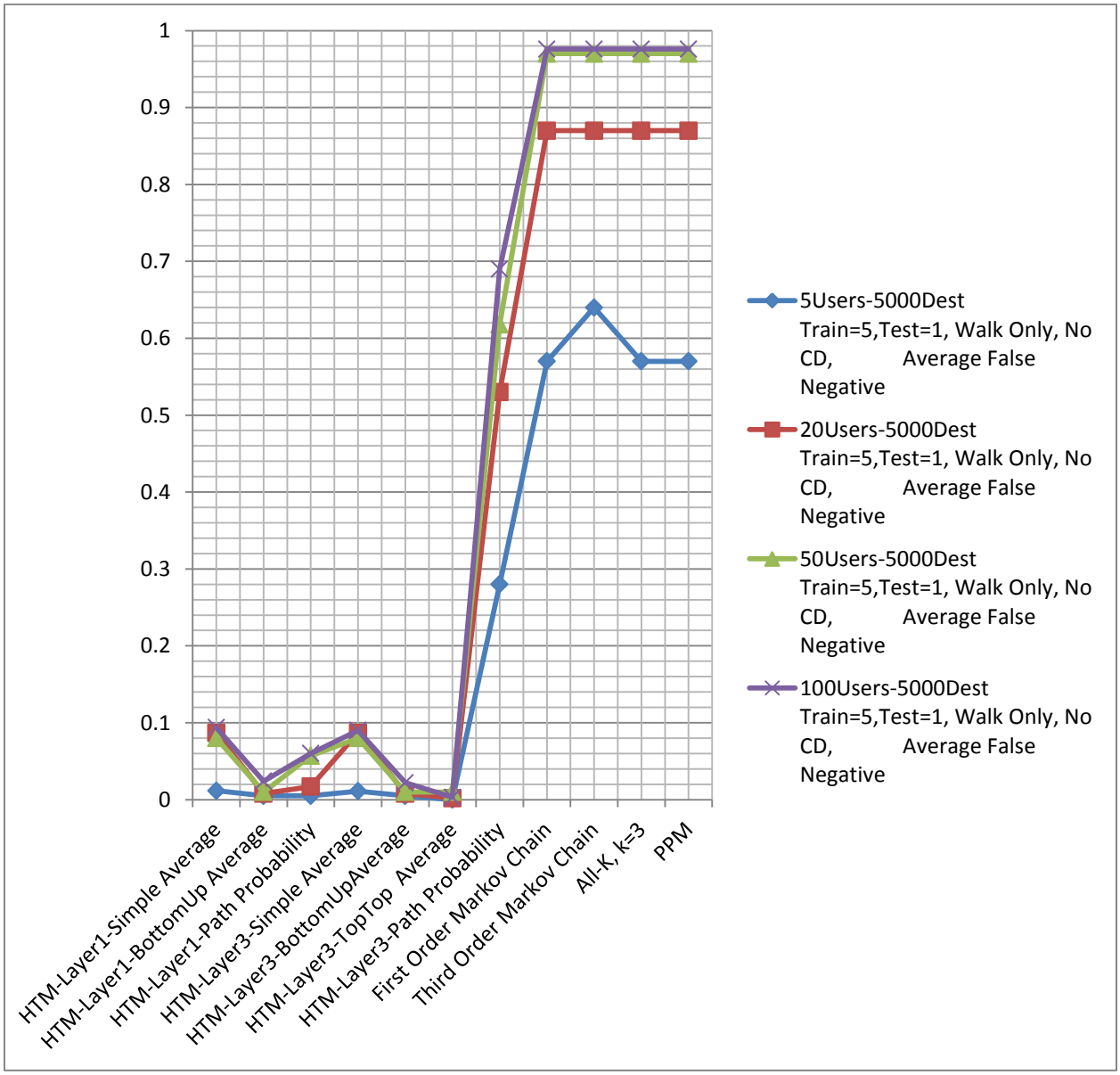


Figure 77 Appendix A Synthetic Data False Negatives Results for 5 Train, 1 Test, 5000 Destinations

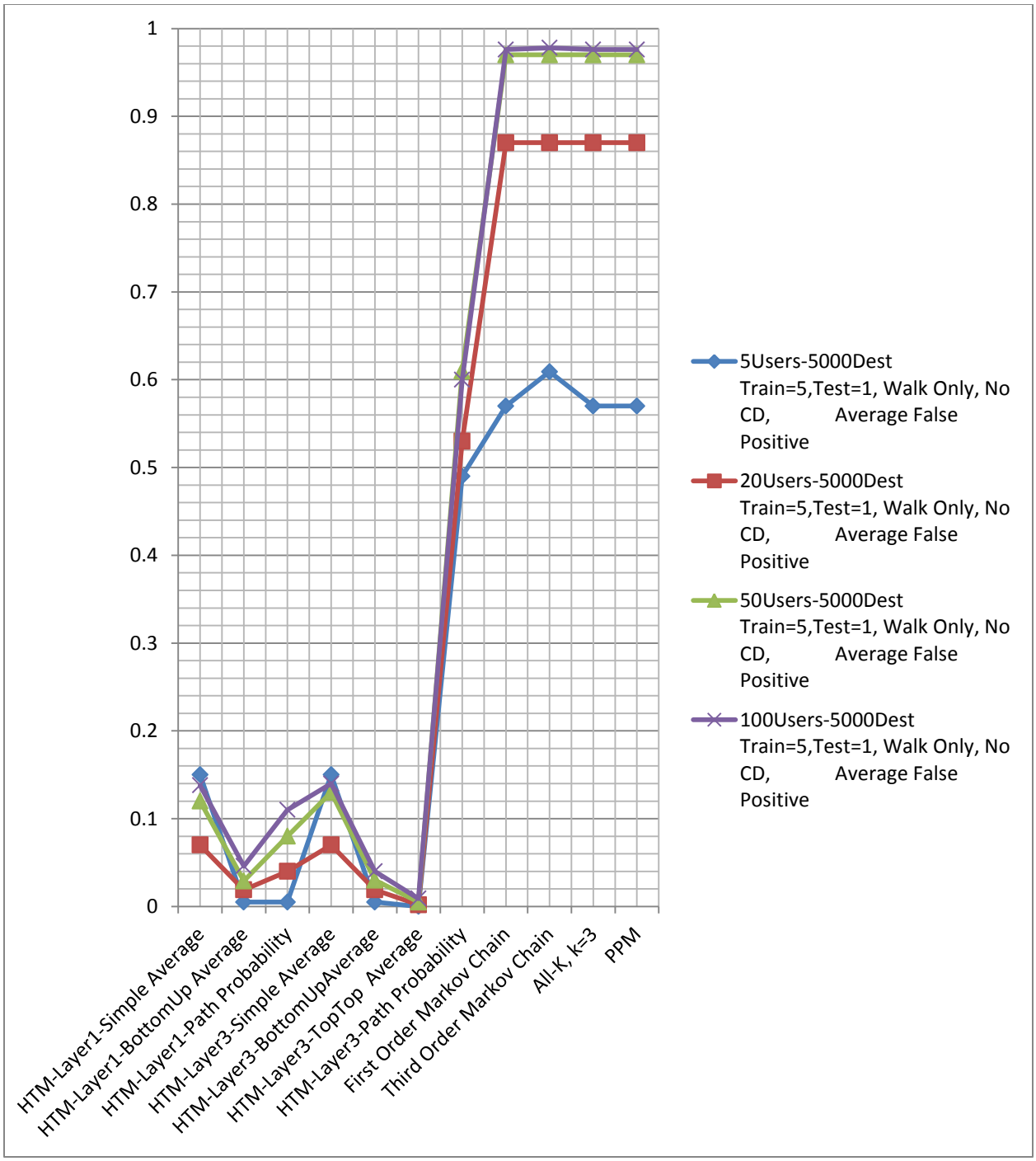


Figure 78 Appendix A Synthetic Data False Positives Results for 5 Train, 1 Test, 5000 Destinations

Five Train days one test day and ten thousand destinations

The graph below supports the precision statistics for 5 train days, 1 test day and 10,000 destinations.

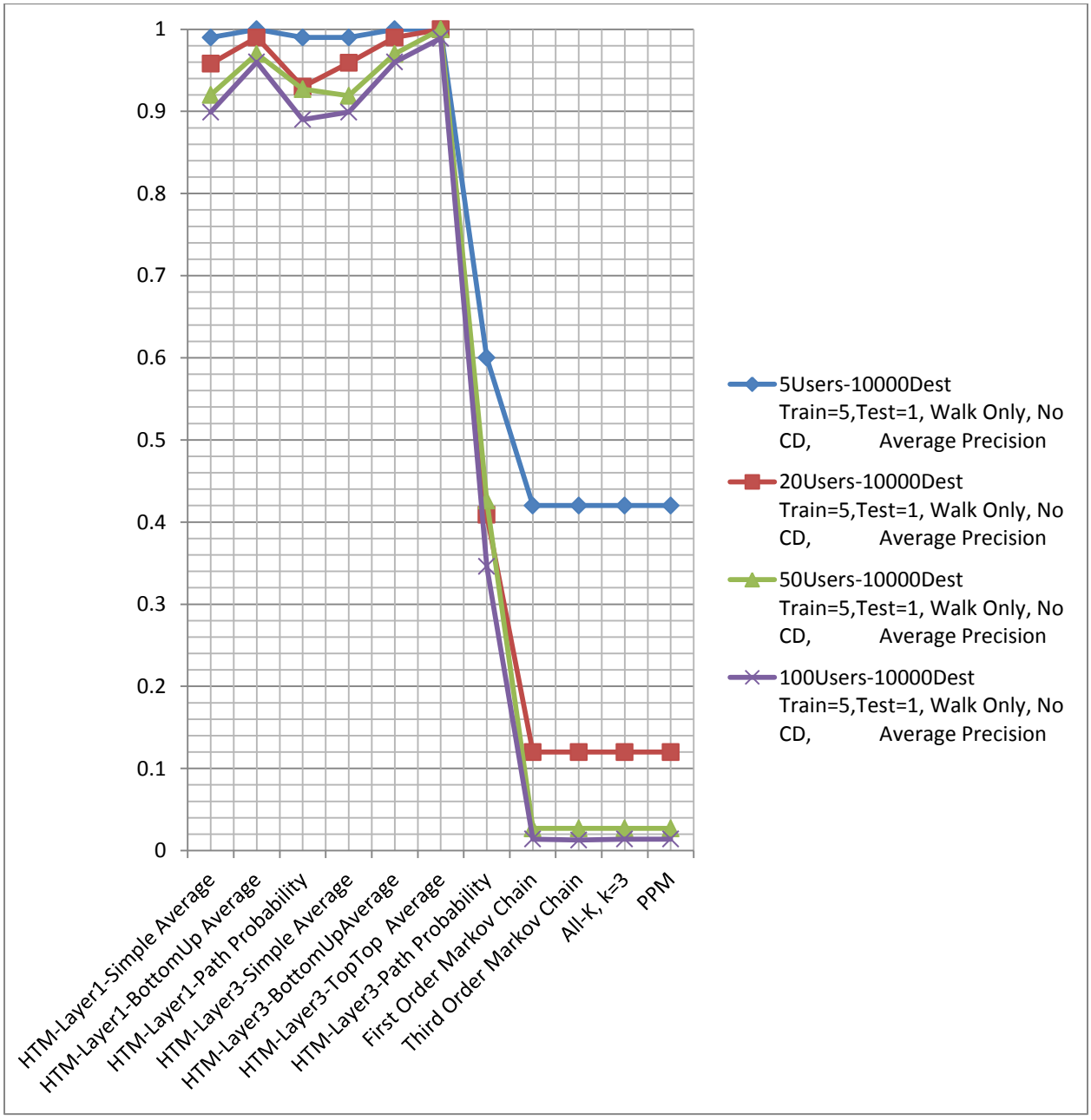


Figure 79 Appendix A Synthetic Data Precision Results for 5 Train, 1 Test, 10,000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 1 test day and 10,000 destinations.

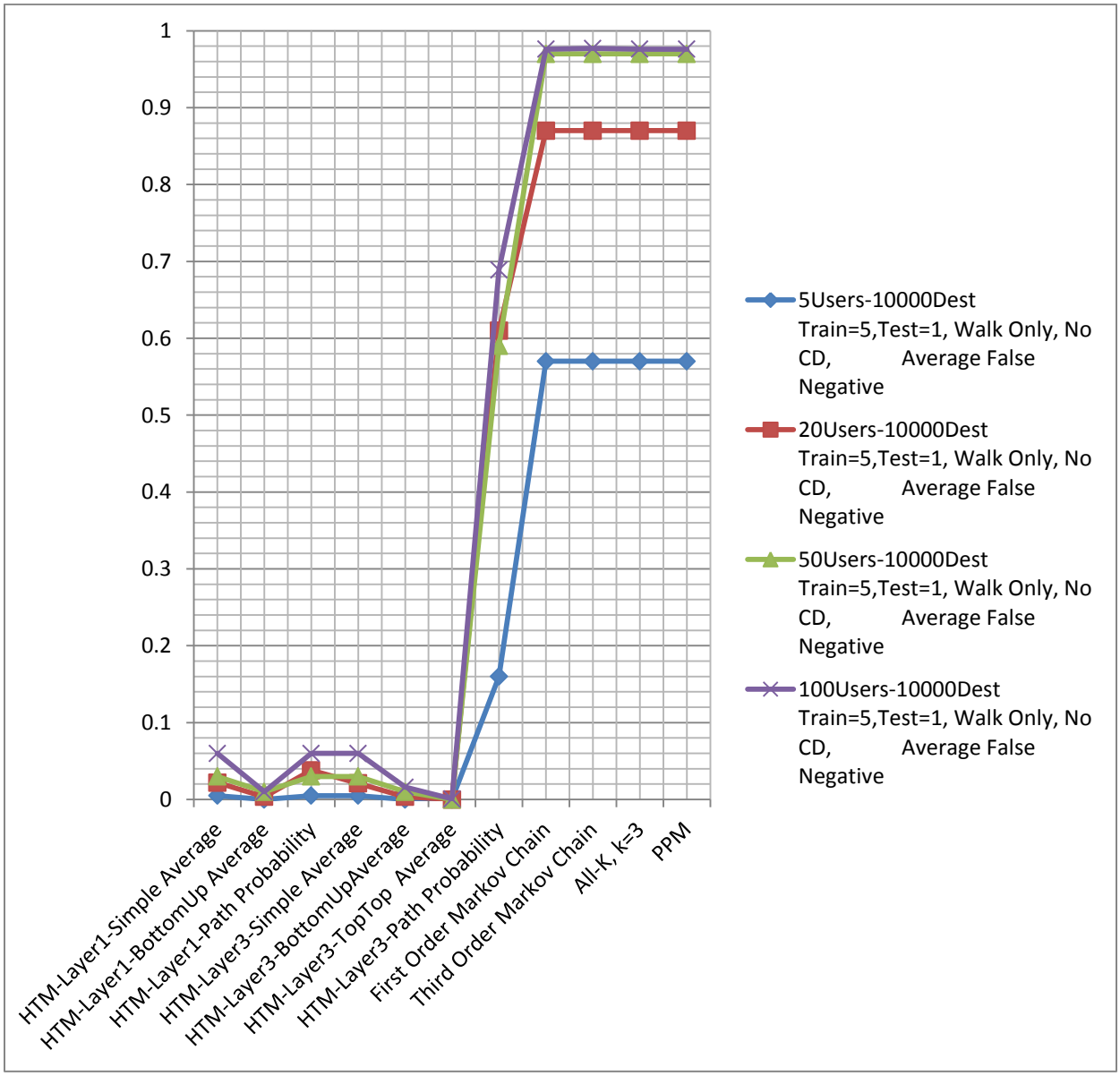


Figure 80 Appendix A Synthetic Data False Negatives Results for 5 Train, 1 Test, 10,000 Destinations

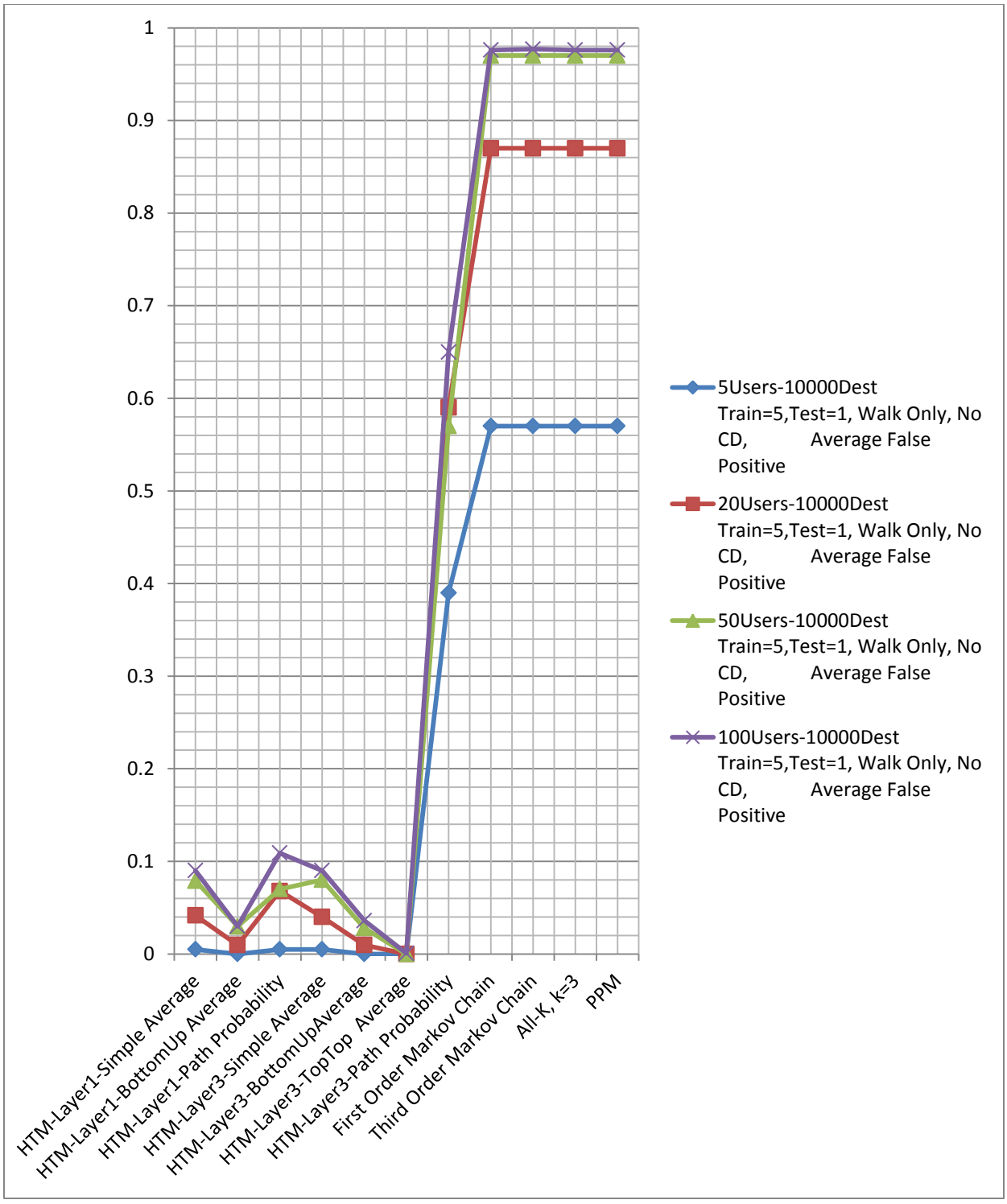


Figure 81 Appendix A Synthetic Data False Positives Results for 5 Train, 1 Test, 10,000 Destinations

Five train days, three observations and one thousand destinations

The graph below supports the precision statistics for 5 train days, three observations and 1000 destinations.

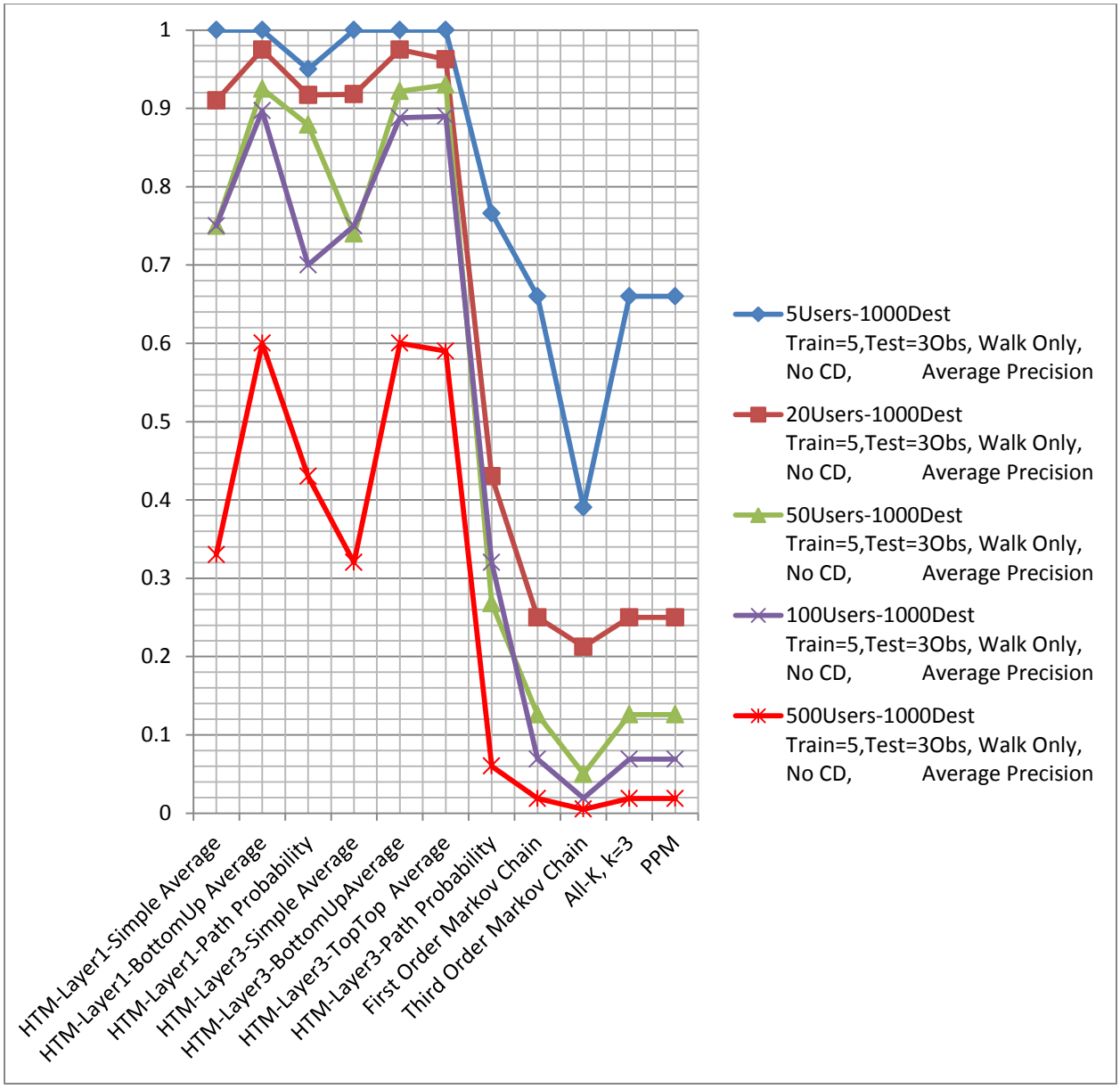


Figure 82 Appendix A Synthetic Data Precision Results for 5 Train, 3 Observations, 1000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 3 observations, and 1000 destinations.

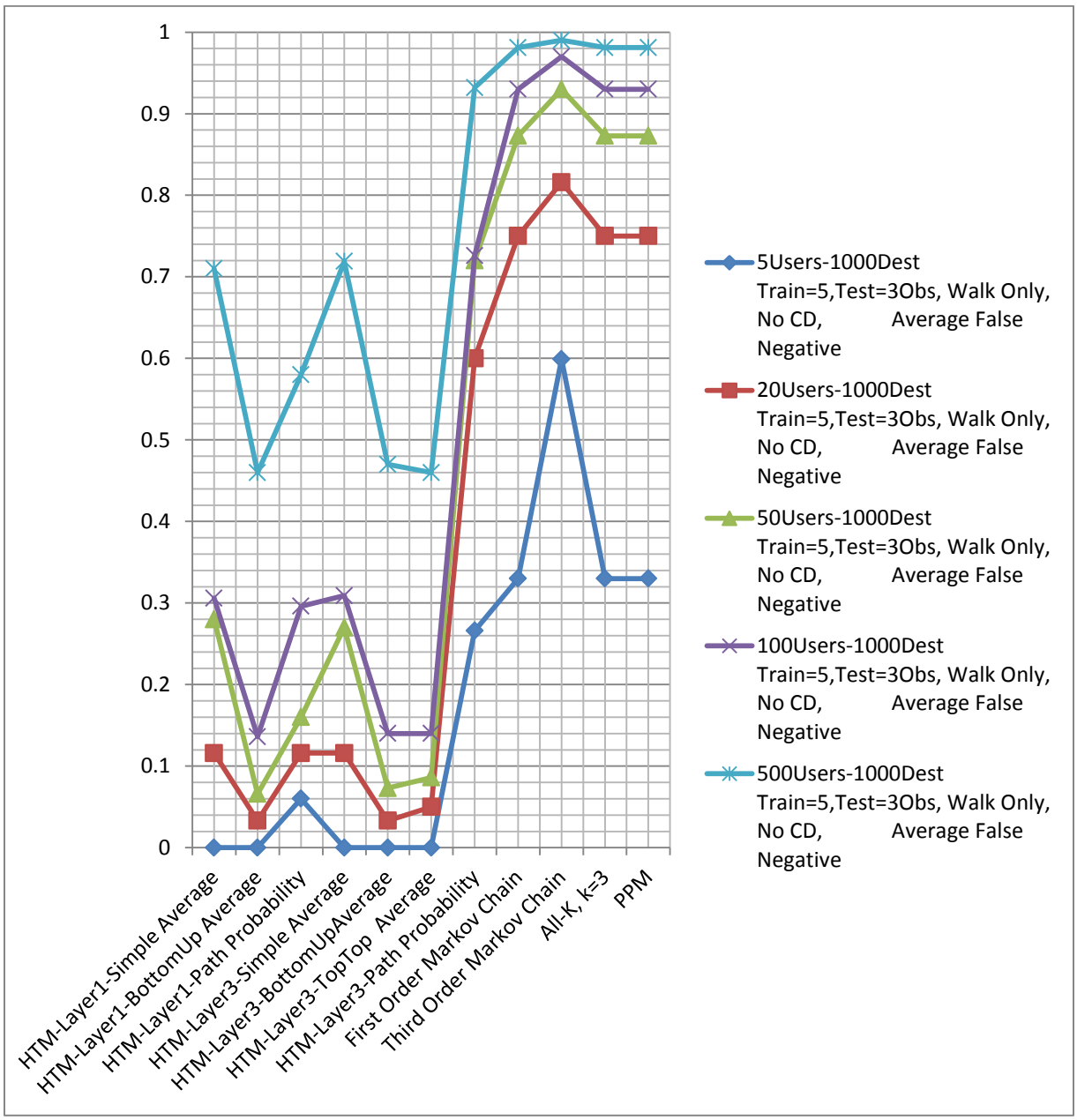


Figure 83 Appendix A Synthetic Data False Negatives Results for 5 Train, 3 Observations, 1000 Destinations

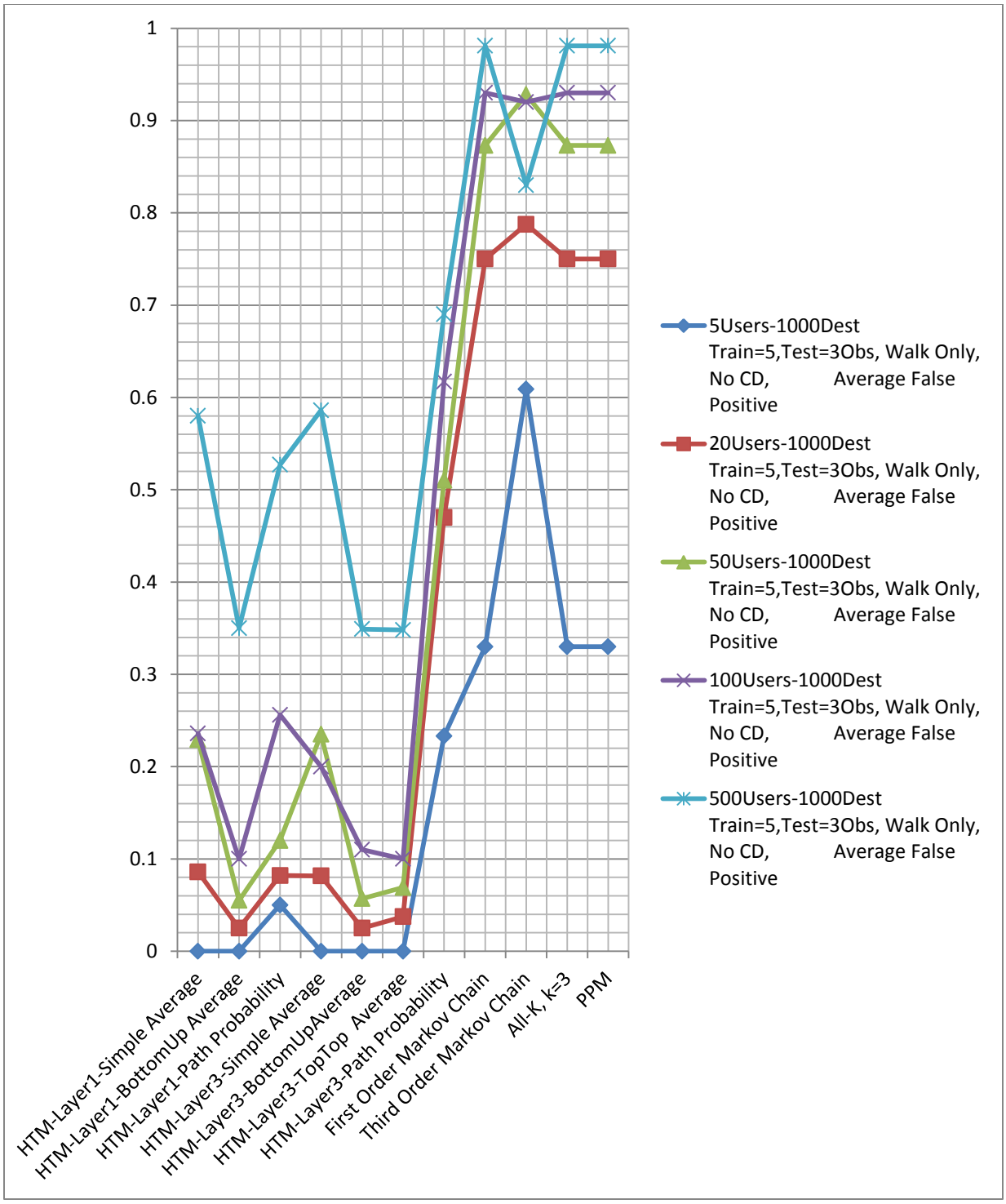


Figure 84 Appendix A Synthetic Data False Positives Results for 5 Train, 3 Observations, 1000 Destinations

Five train days, three observations and five thousand destinations

The graph below supports the precision statistics for 5 train days, three observations and 5000 destinations.

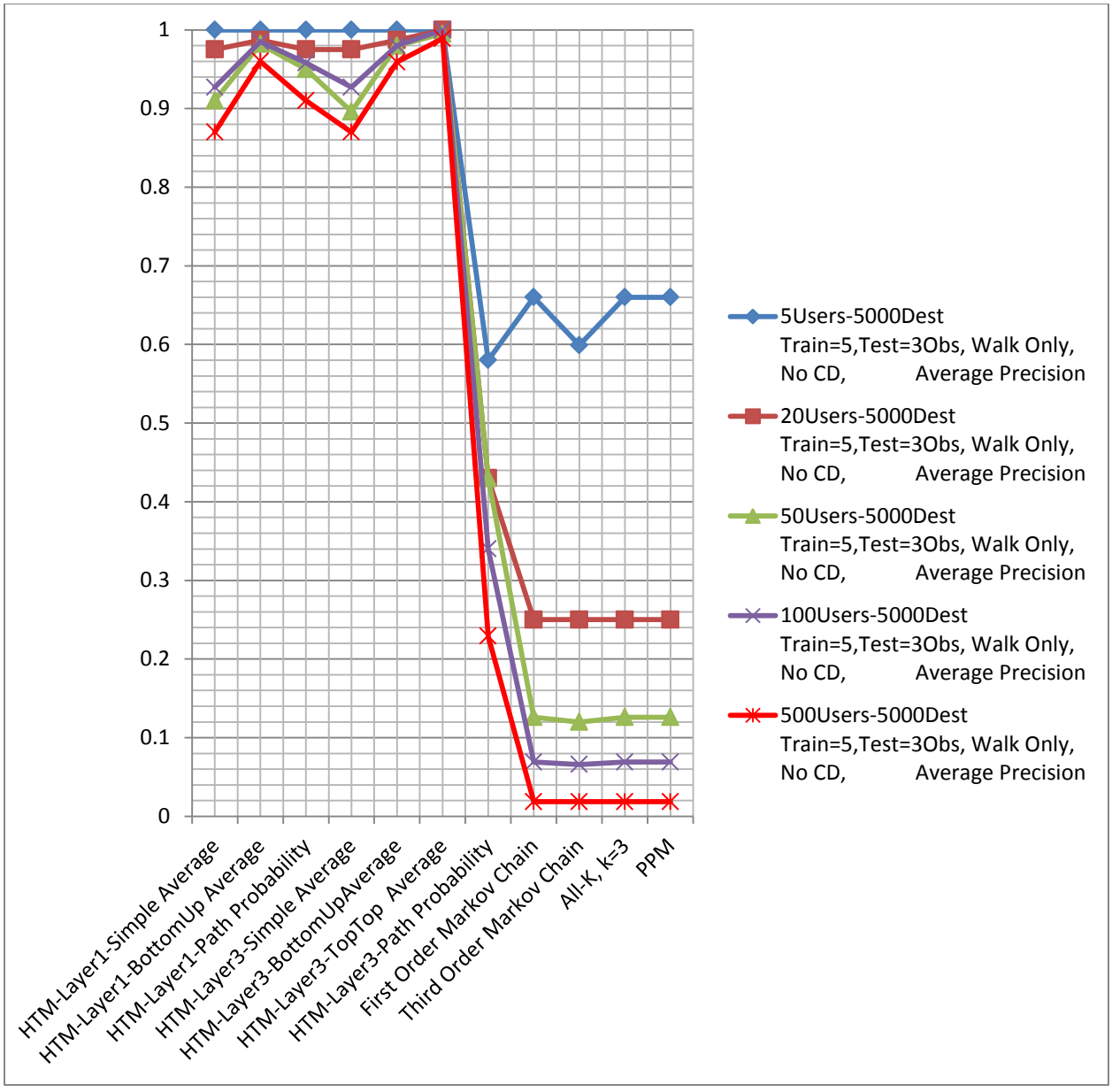


Figure 85 Appendix A Synthetic Data Precision Results for 5 Train, 3 Observations, 5000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 3 observations, and 5000 destinations.

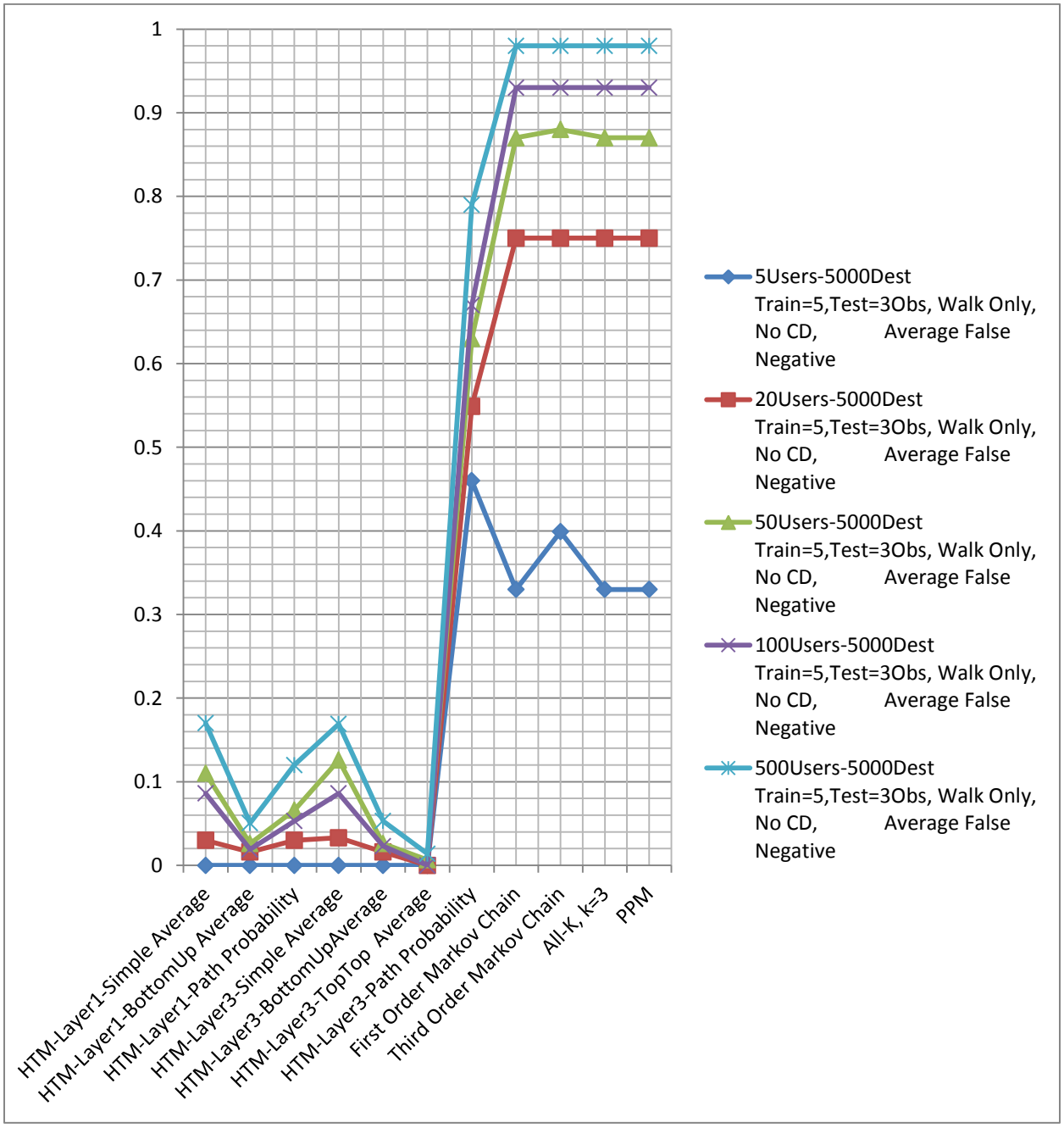


Figure 86 Appendix A Synthetic Data False Negatives Results for 5 Train, 3 Observations, 5000 Destinations

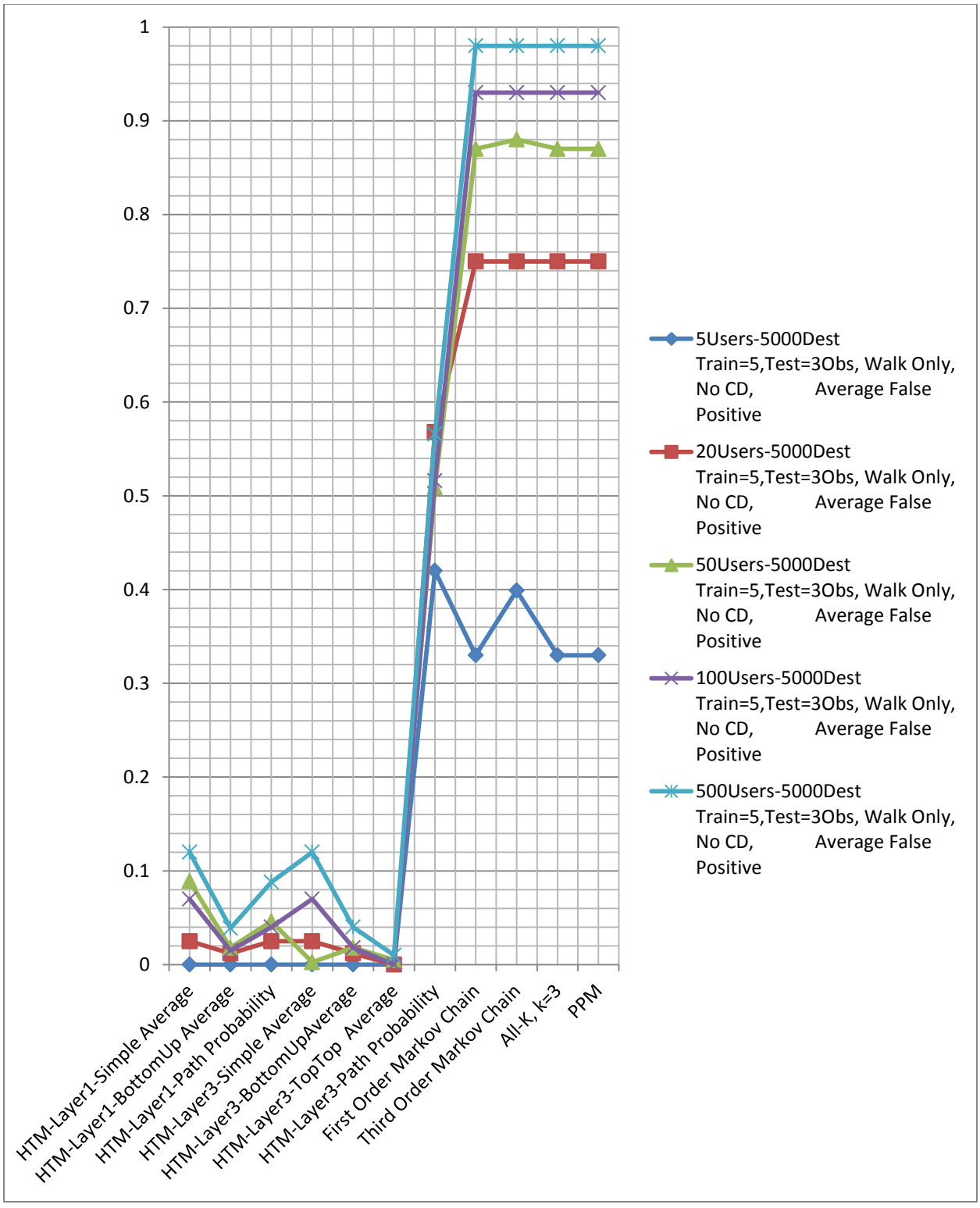


Figure 87 Appendix A Synthetic Data False Positives Results for 5 Train, 3 Observations, 5000 Destinations

Five train days, three observations and ten thousand destinations

The graph below supports the precision statistics for 5 train days, three observations and 10,000 destinations.

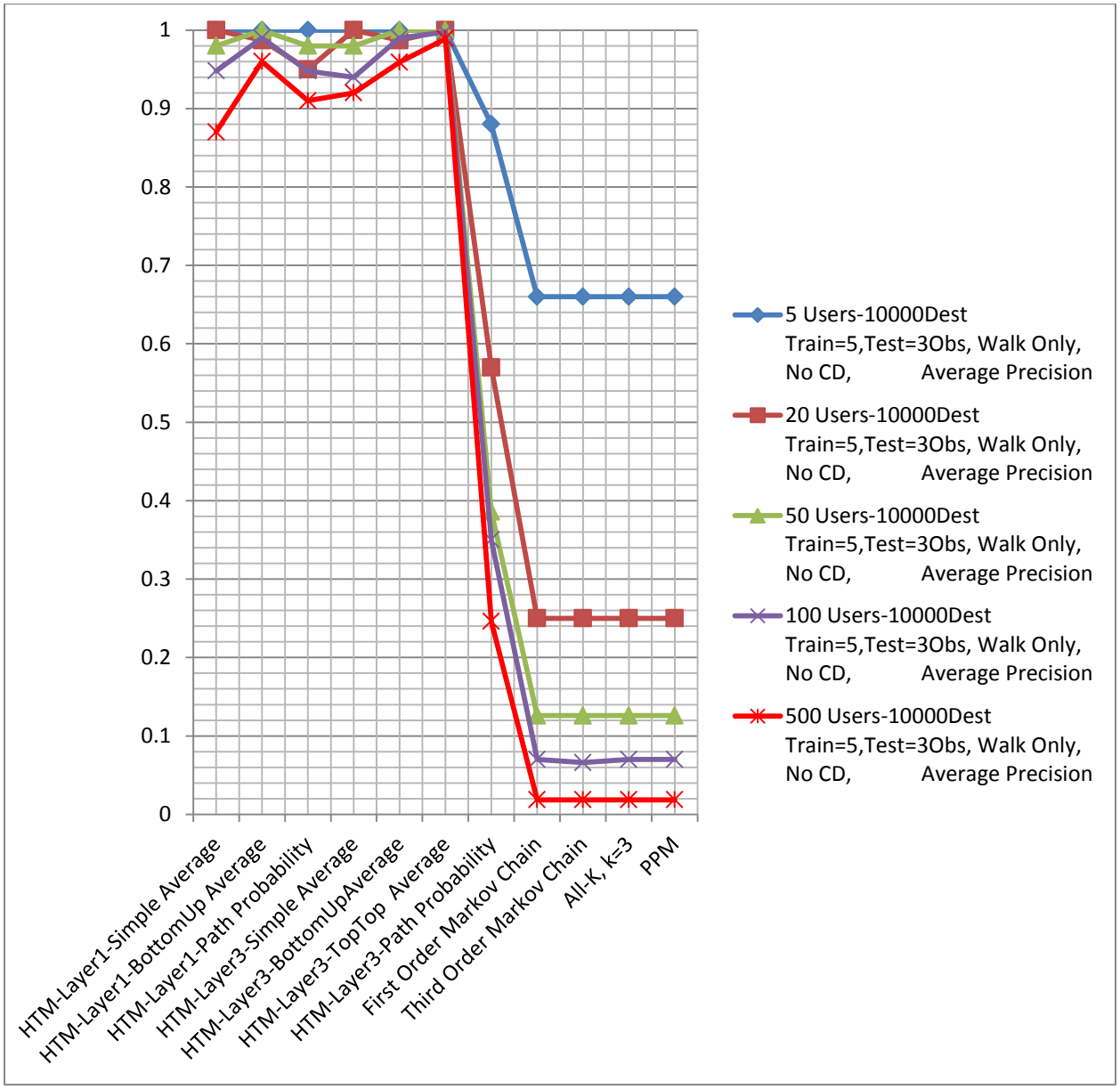


Figure 88 Appendix A Synthetic Data Precision Results for 5 Train, 3 Observations, 10,000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 3 observations, and 10,000 destinations.

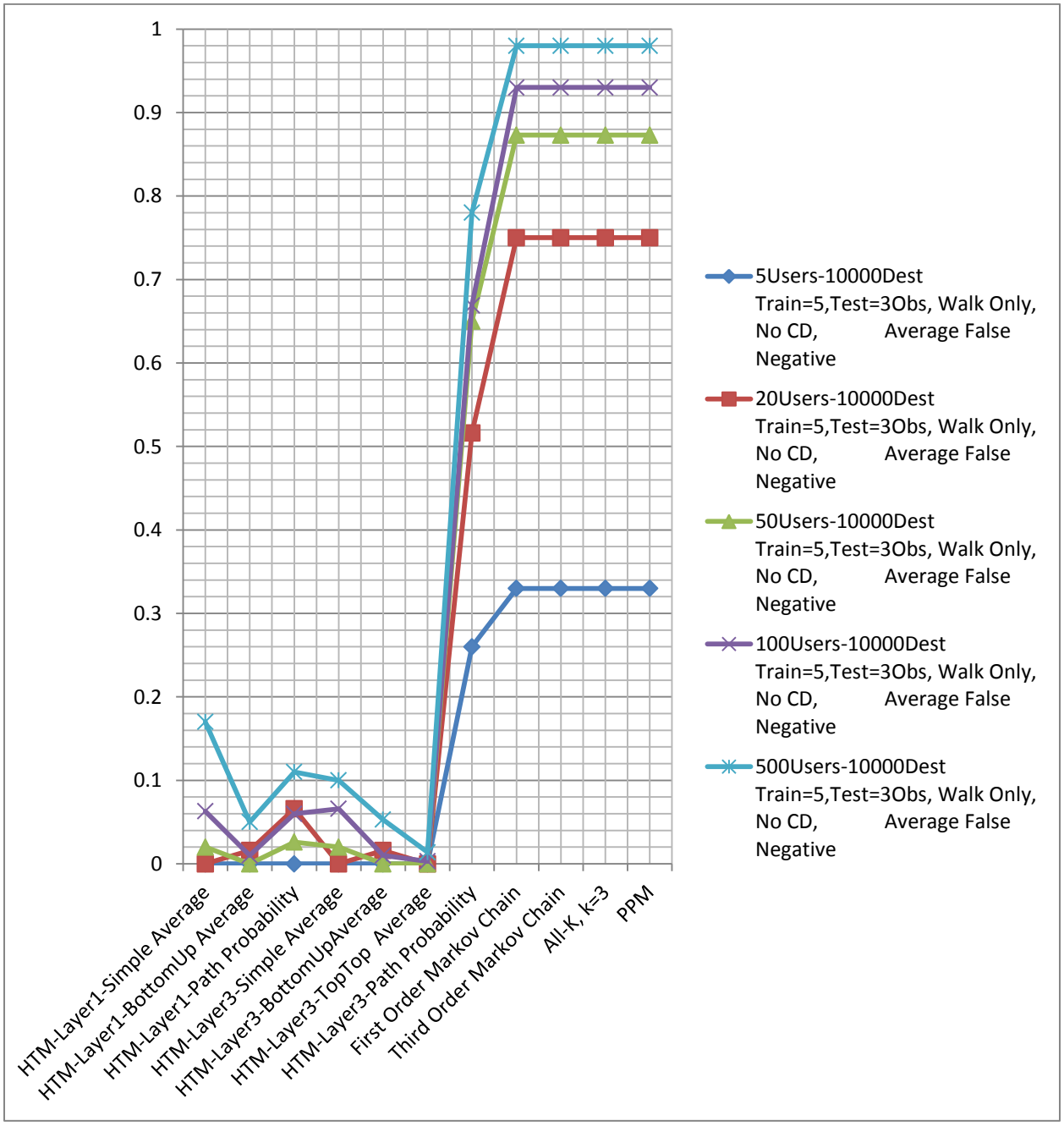


Figure 89 Appendix A Synthetic Data False Negative Results for 5 Train, 3 Observations, 10,000 Destinations

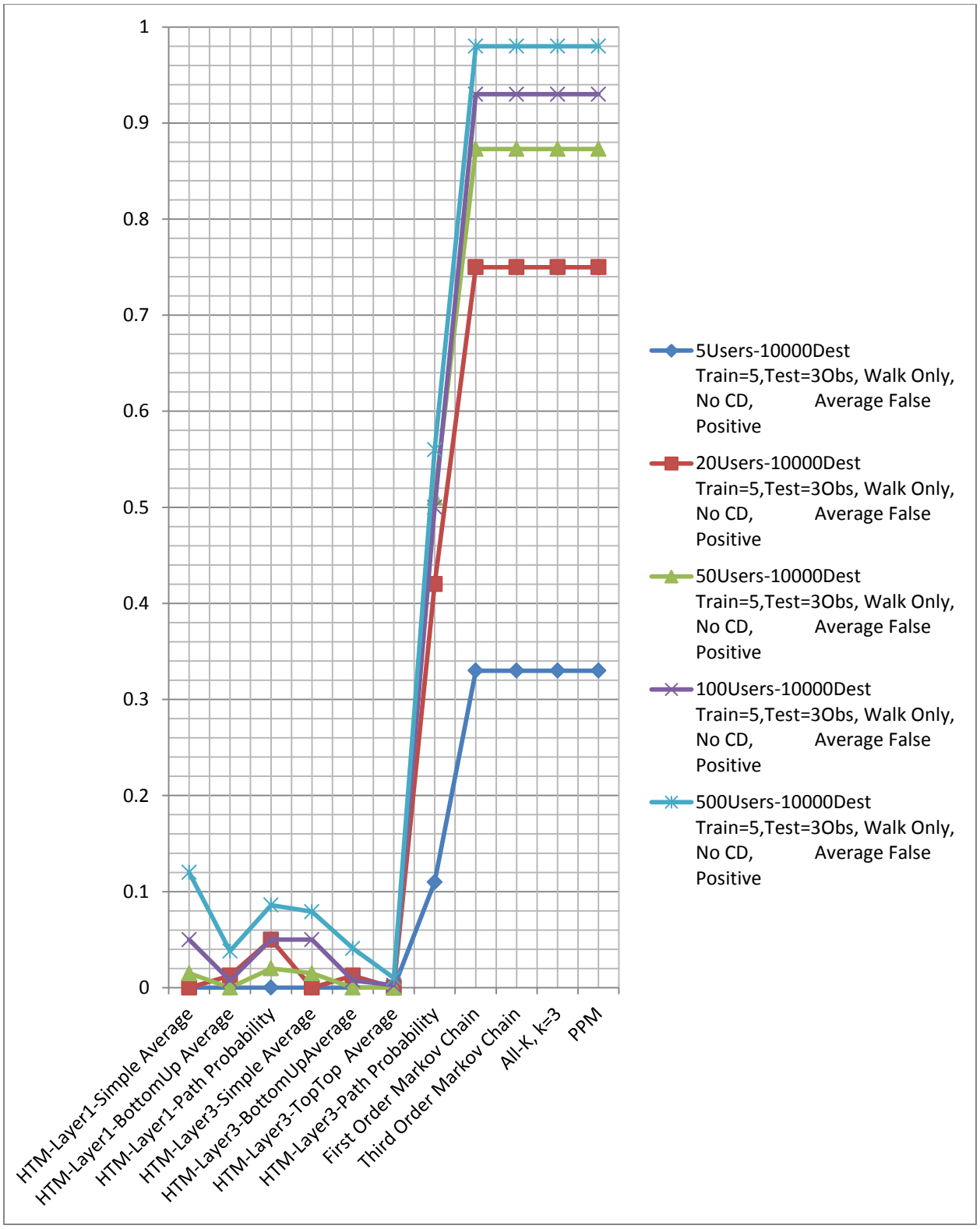


Figure 90 Appendix A Synthetic Data False Positives Results for 5 Train, 3 Observations, 10,000 Destinations

Appendix B

User Attribution Test Results Using Synthetic Data with Concept Drift

Five train days, two test day, one thousand destinations and five users

The graph below supports the precision statistics for 5 train days, two test days and 1000 destinations.

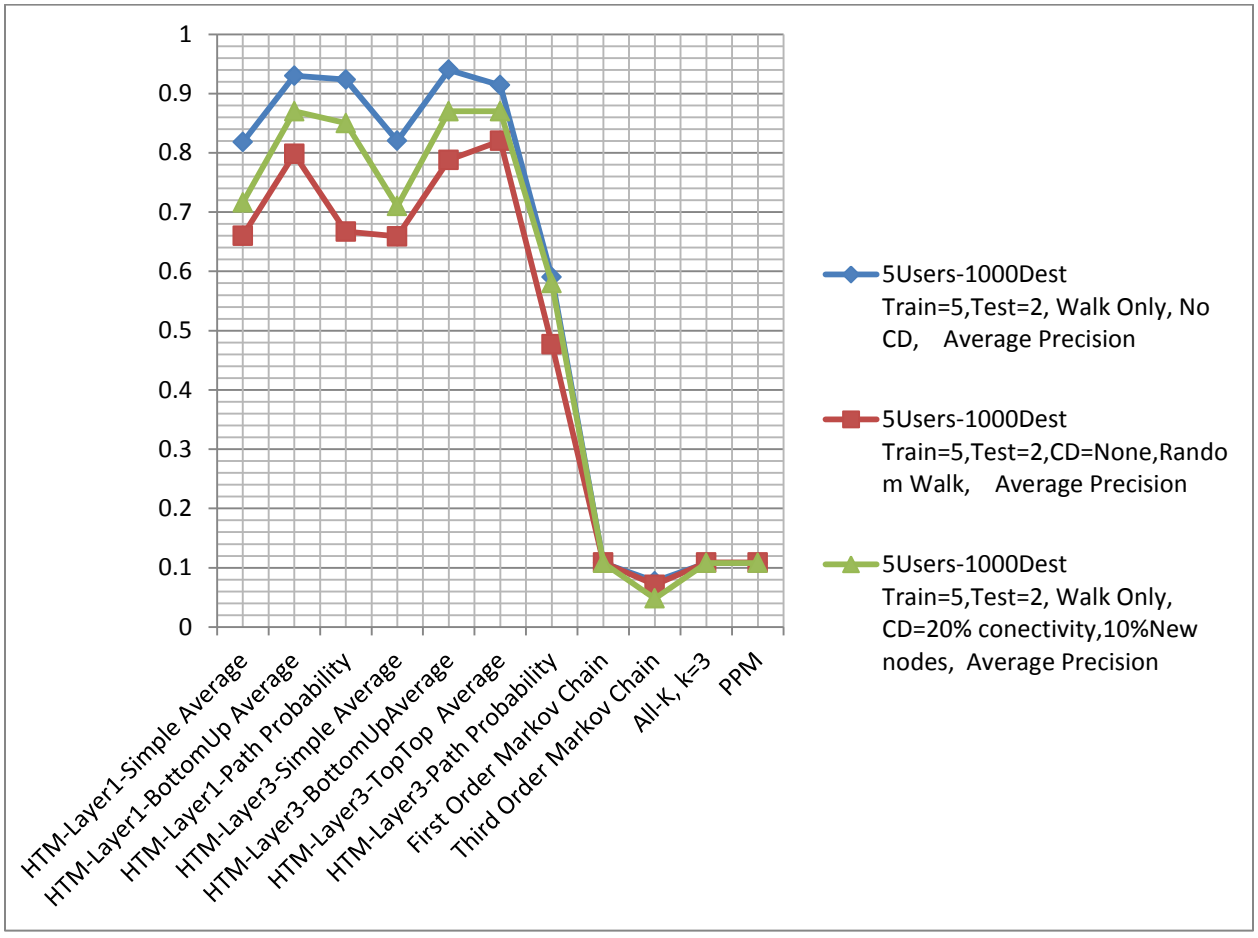


Figure 91 Appendix B Synthetic Data Precision Results with Concept Drift 5 Train, 2 Test, 1000 Destinations

The next graphs report false negative and false positive statistics for 5 train days, 2 test days, and 1000 destinations.

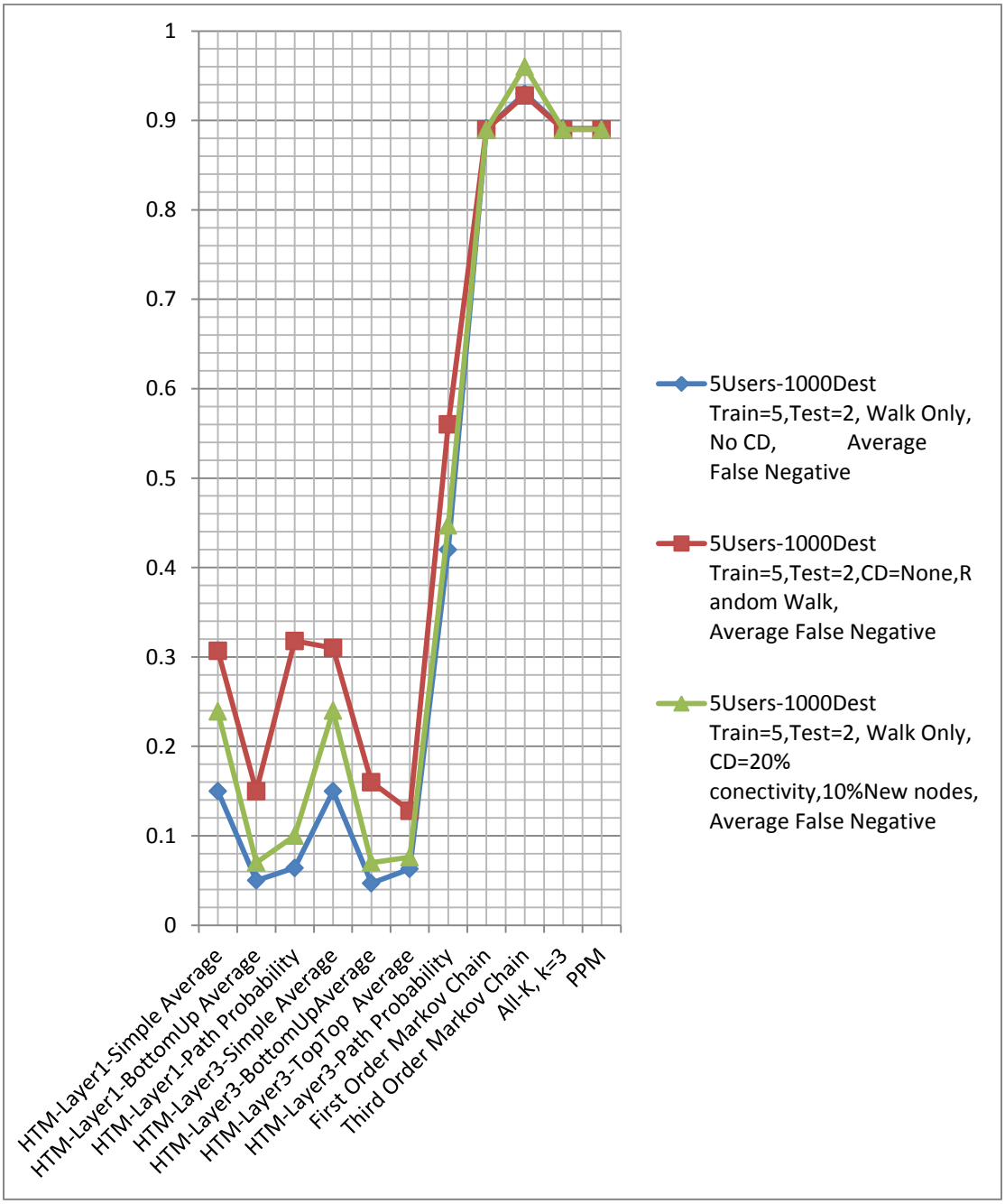


Figure 92 Appendix B Synthetic Data False Negatives Results with Concept Drift 5 Train, 2 Test, 1000 Destinations

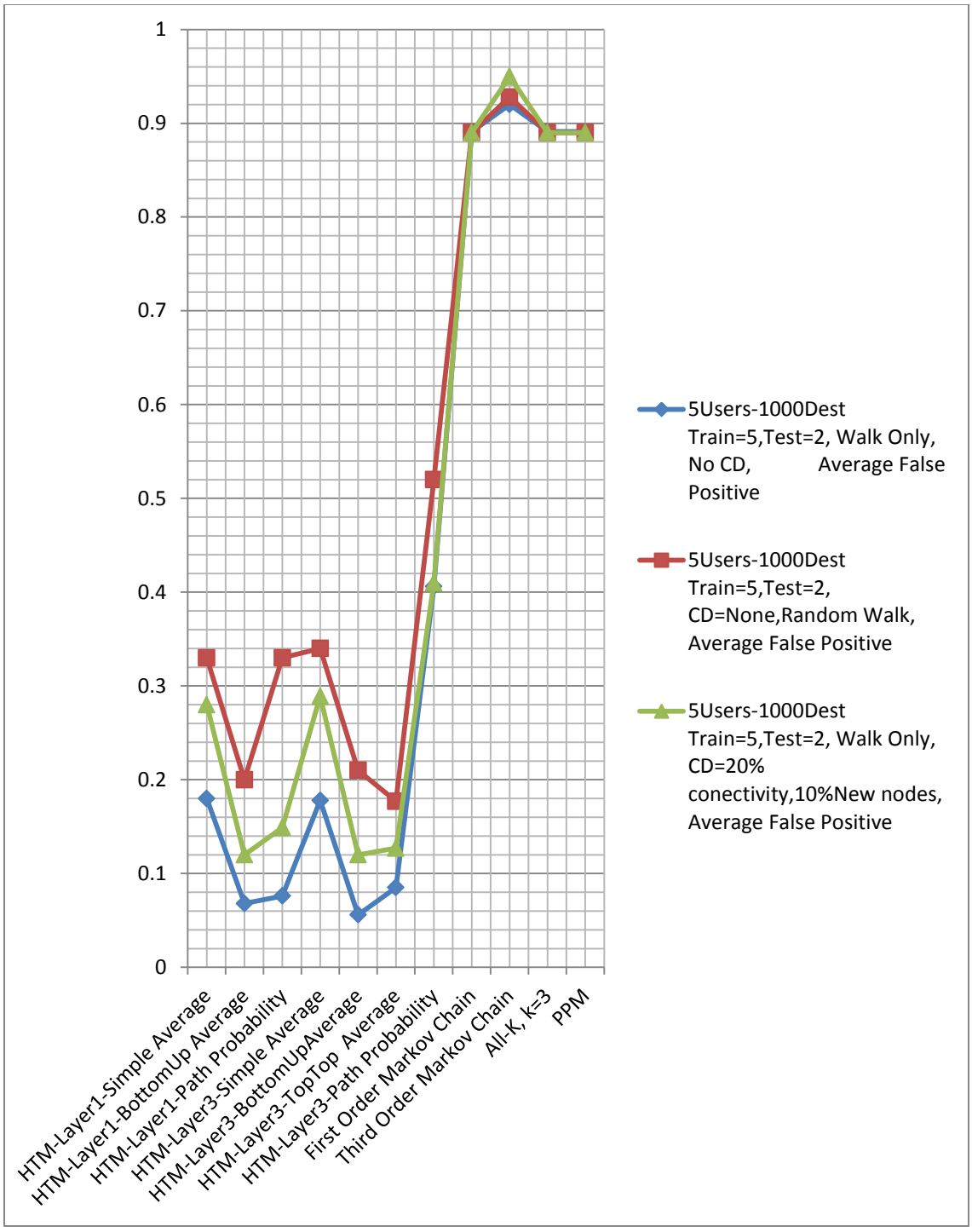


Figure 93 Appendix B Synthetic Data False Positives Results with Concept Drift 5 Train, 2 Test, 1000 Destinations

Ten train days, three test day, one thousand destinations and five users

The graph below supports the recall statistics for 10 train days, three test days and 1000 destinations.

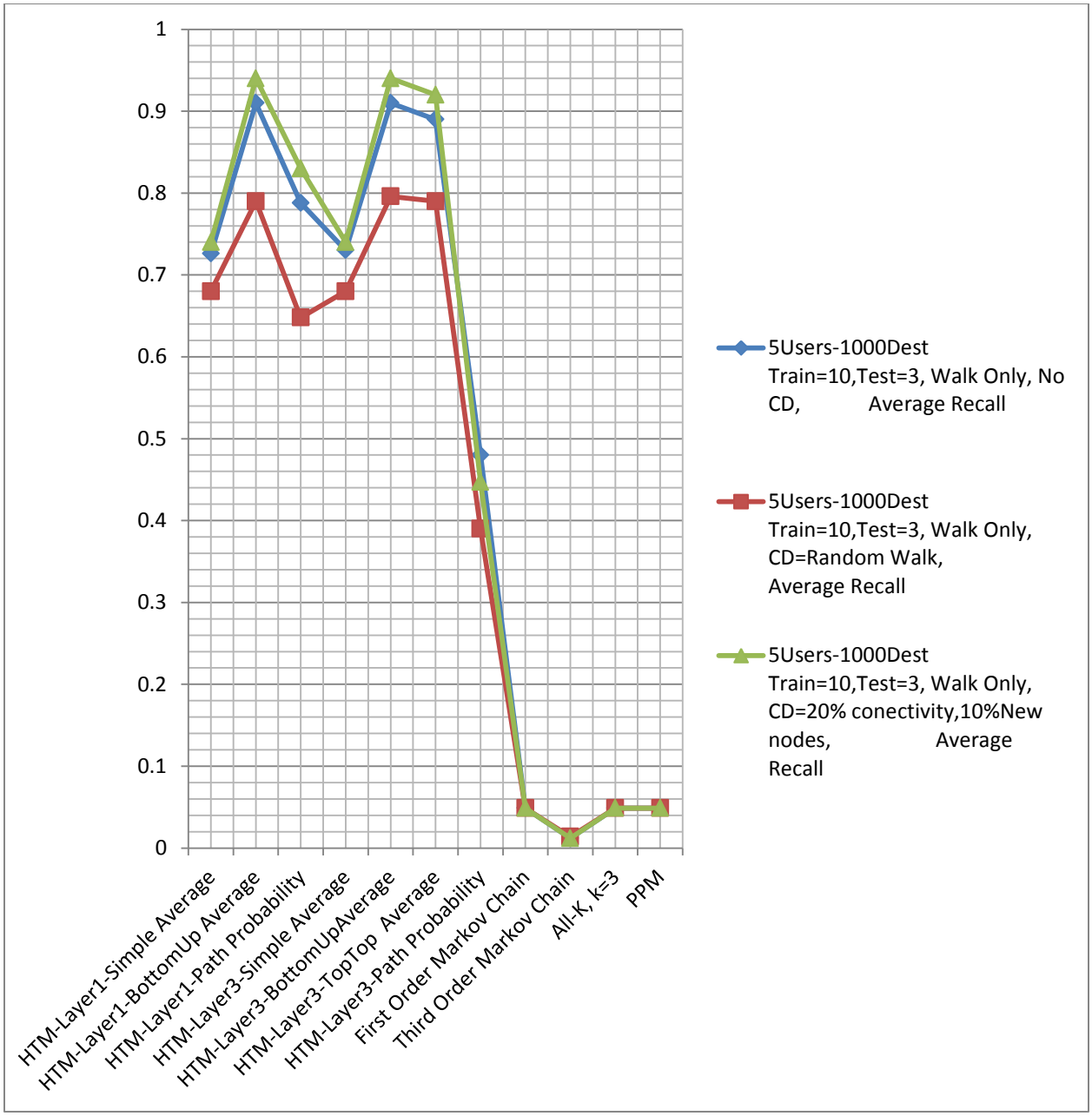


Figure 94 Appendix B Synthetic Data Recall Results with Concept Drift 10 Train, 3 Test, 1000 Destinations

The graph below supports the precision statistics for 10 train days, three test days and 1000 destinations.

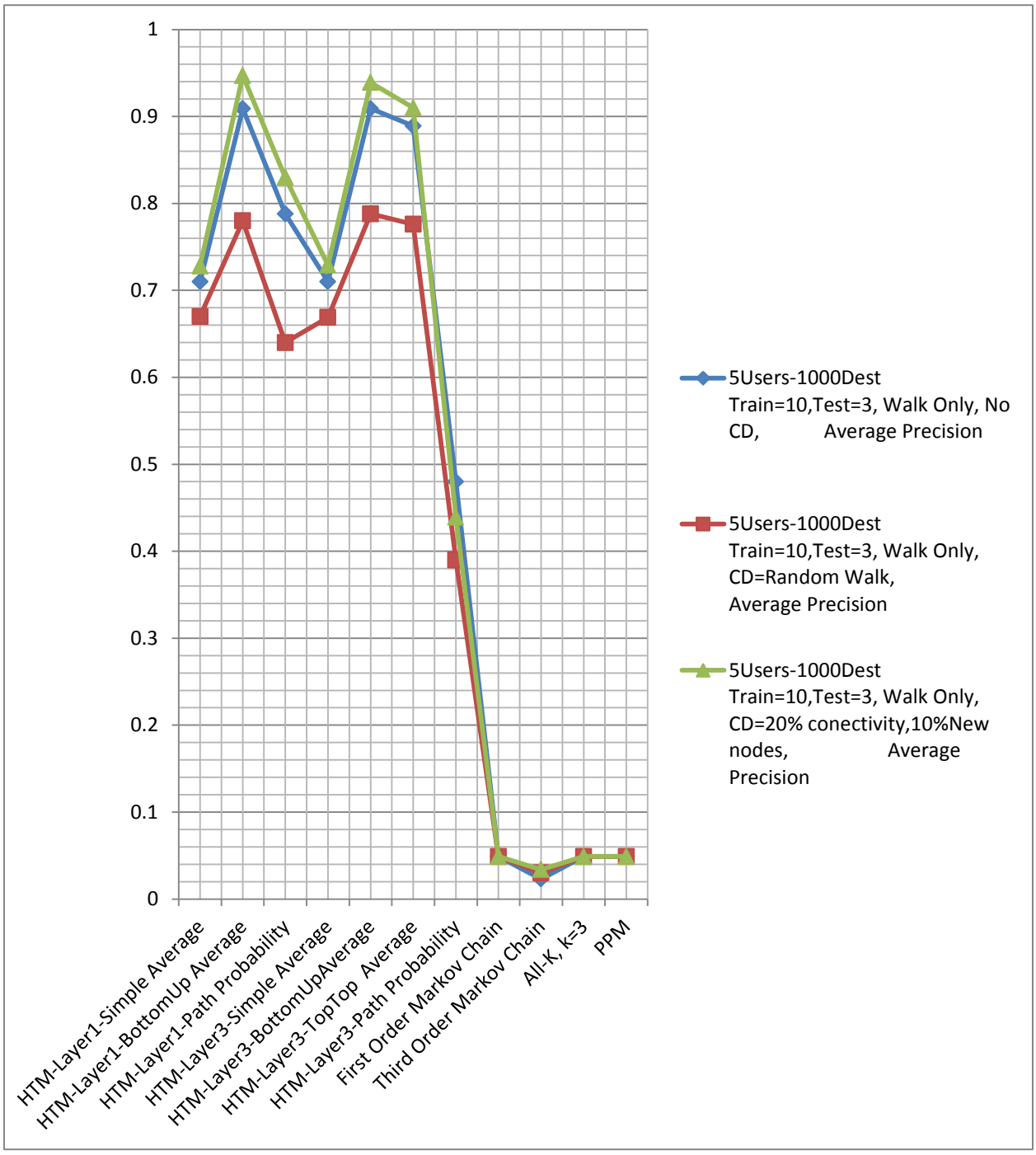


Figure 95 Appendix B Synthetic Data Precision Results with Concept Drift 10 Train, 3 Test, 1000 Destinations

The next graphs report false negative and false positive statistics for 10 train days, 3 test days, and 1000 destinations.

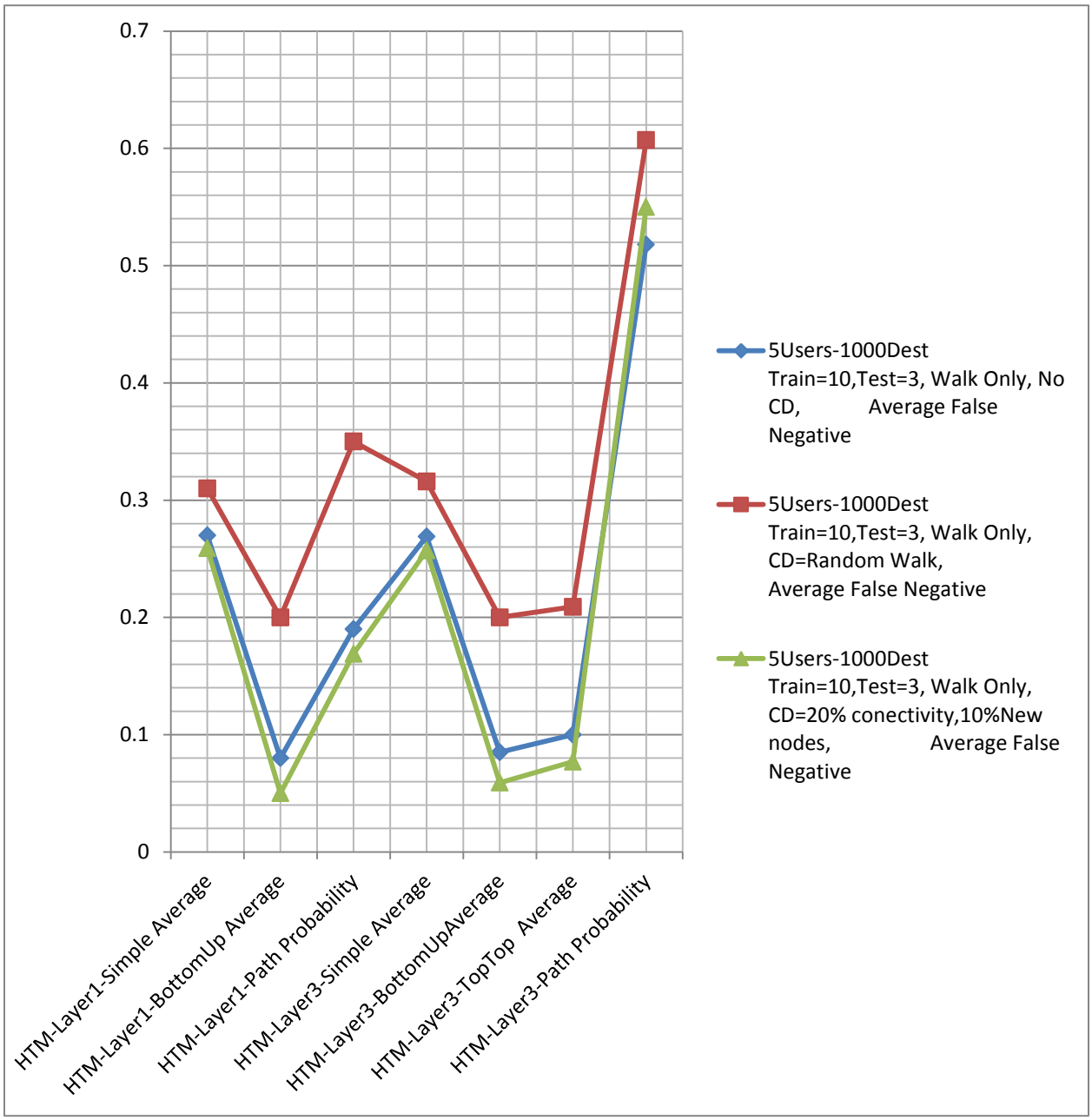


Figure 96 Appendix B Synthetic Data False Negatives Results with Concept Drift 10 Train, 3 Test, 1000 Destinations

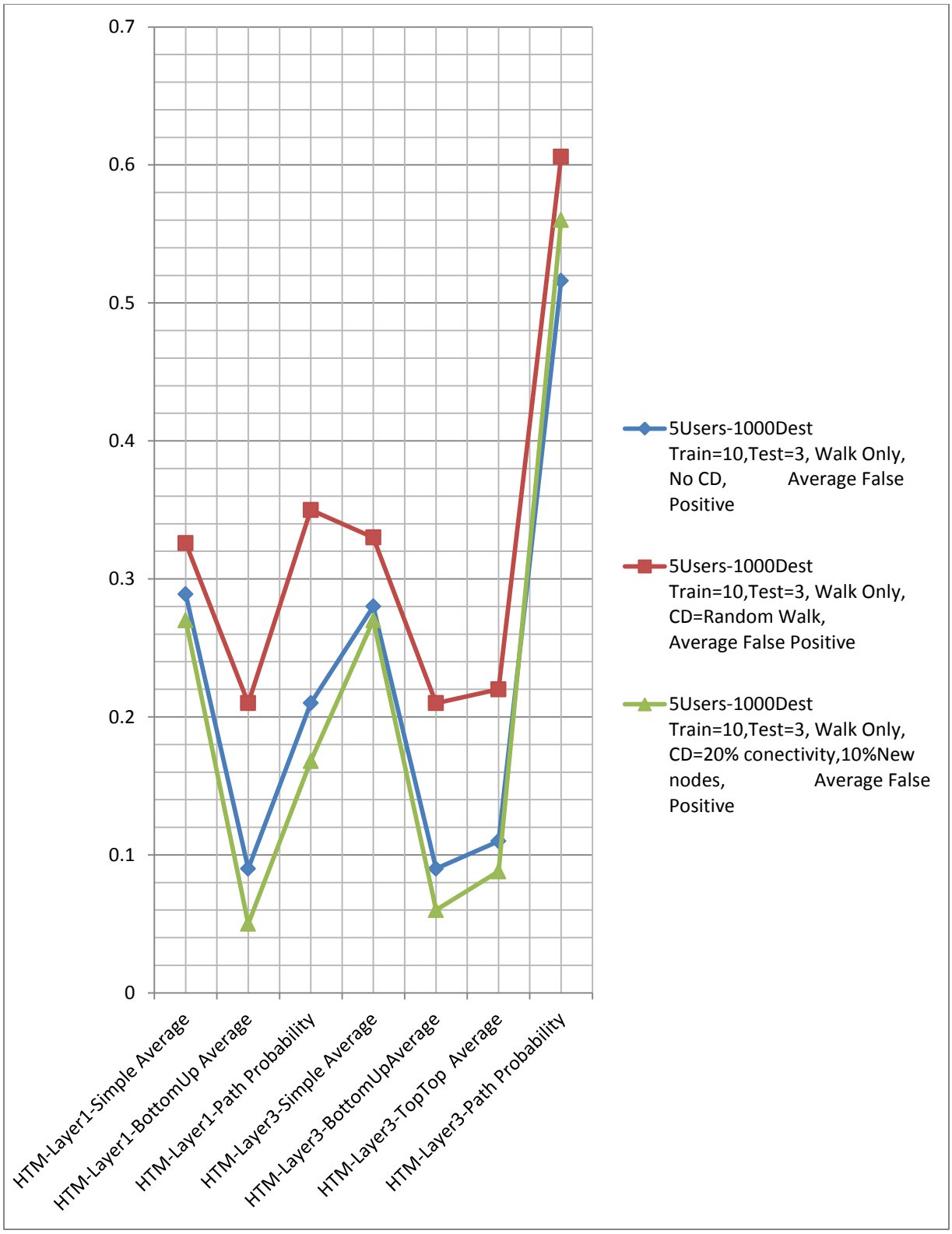


Figure 97 Appendix B Synthetic Data False Positives Results with Concept Drift 10 Train, 3 Test, 1000 Destinations

Recall difference between the baseline (at zero) and the random walk and context drift algorithms.

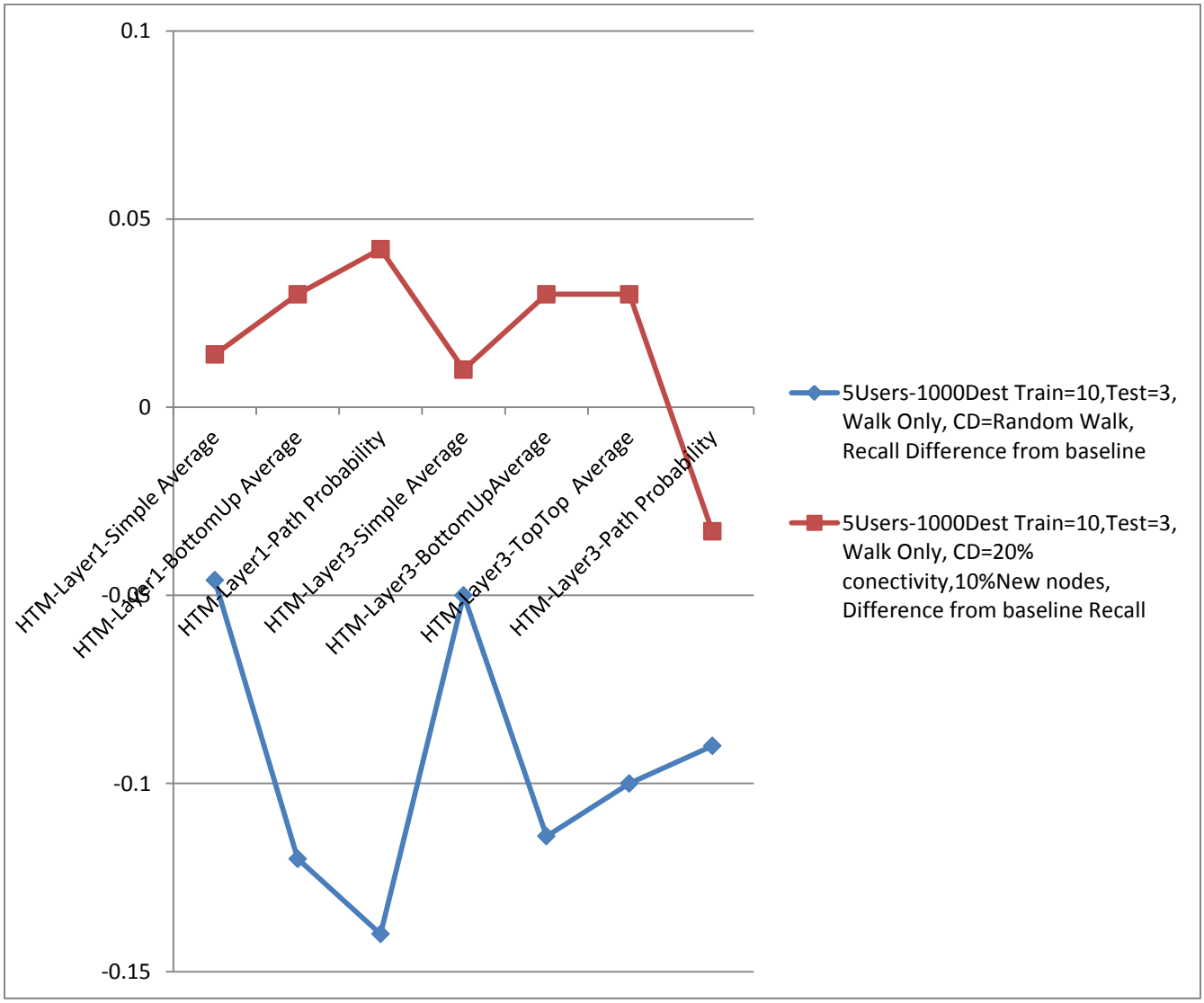


Figure 98 Appendix B Recall difference between baseline and Concept Drift for 10 Train, 3 Test, 1000 Destinations

Fifteen train days, four test day, one thousand destinations and five users

The graph below supports the recall statistics for 15 train days, 4 test days and 1000 destinations.

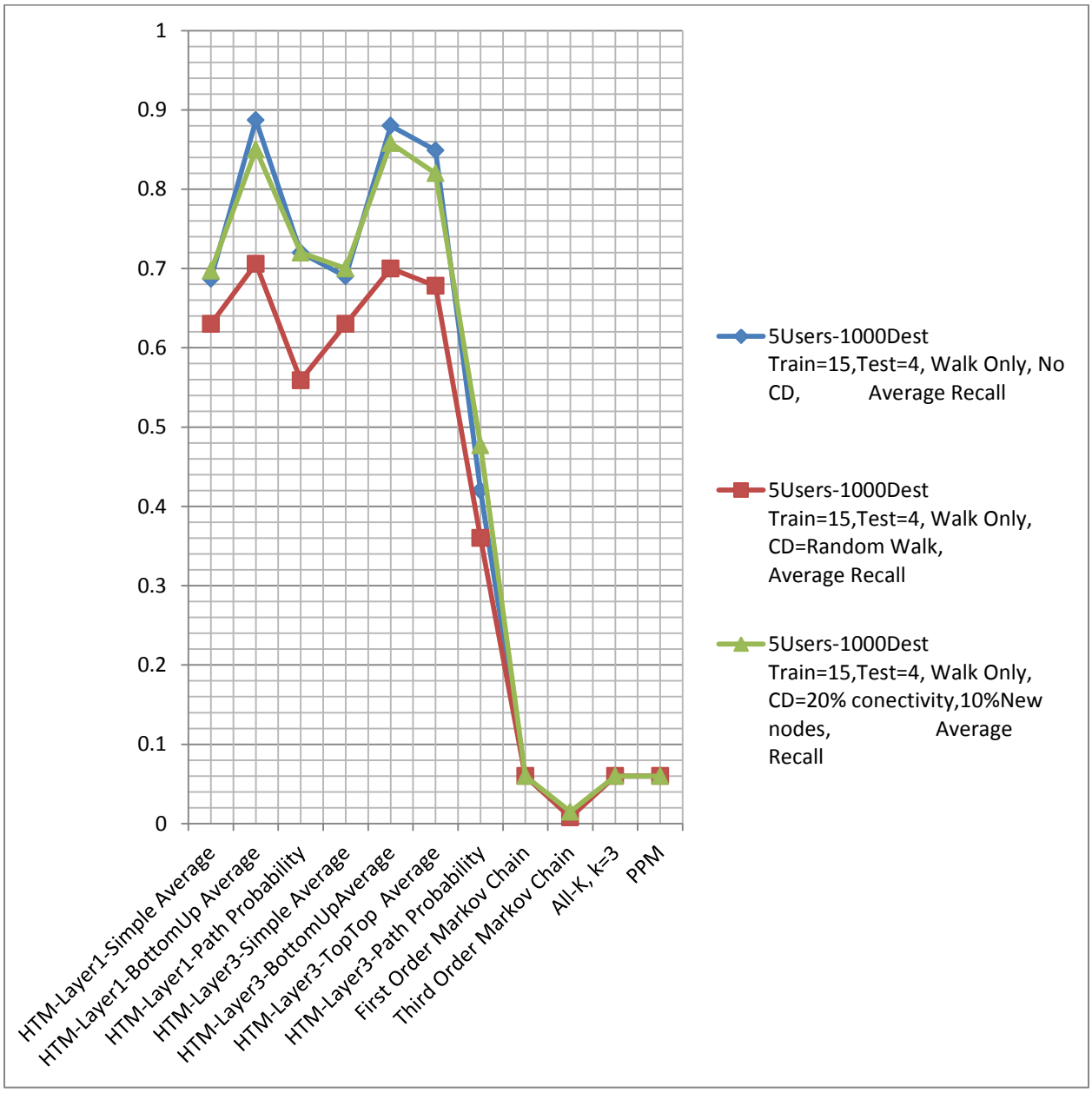


Figure 99 Appendix B Synthetic Data Recall Results with Concept Drift 15 Train, 4 Test, 1000 Destinations

The graph below supports the precision statistics for 15 train days, 4 test days and 1000 destinations.

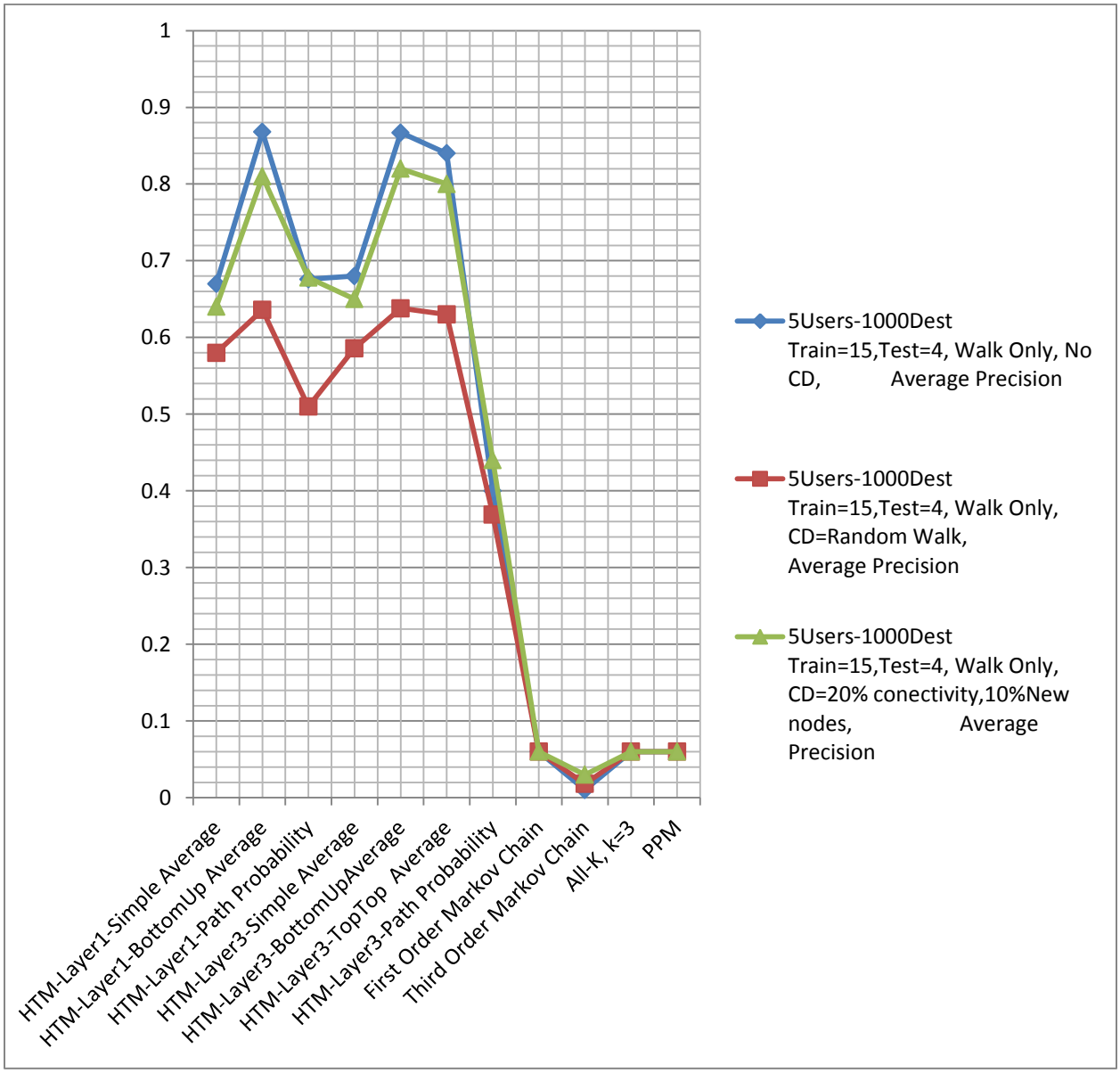


Figure 100 Appendix B Synthetic Data Precision Results with Concept Drift 15 Train, 4 Test, 1000 Destinations

The next graphs report false negative and false positive statistics for 15 train days, 4 test days, and 1000 destinations

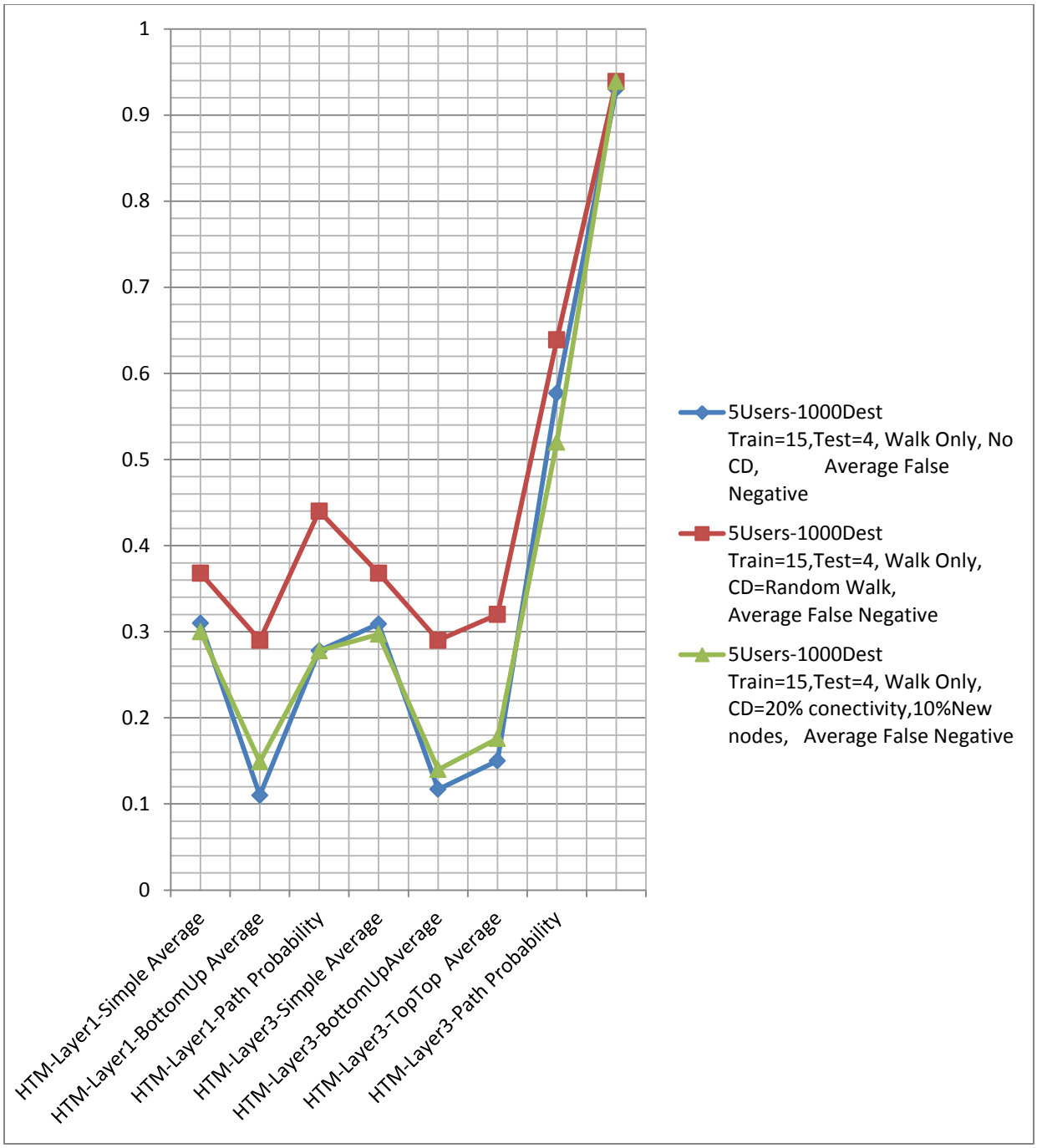


Figure 101 Appendix B Synthetic Data False Negatives Results with Concept Drift 15 Train, 4 Test, 1000 Destinations

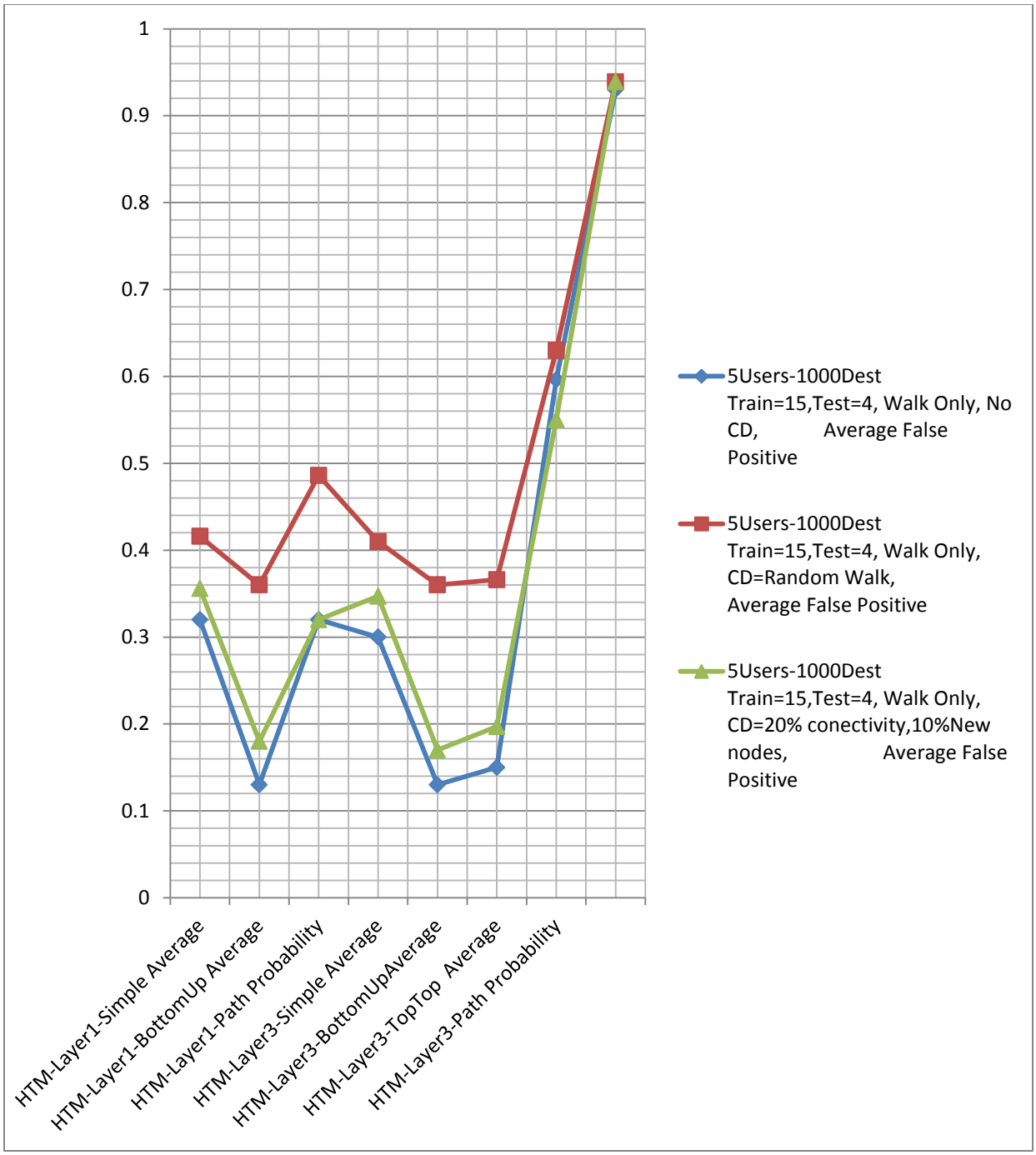


Figure 102 Appendix B Synthetic Data False Positives Results with Concept Drift 15 Train, 4 Test, 1000 Destinations

Recall difference between the baseline (at zero) and the random walk and context drift algorithms.

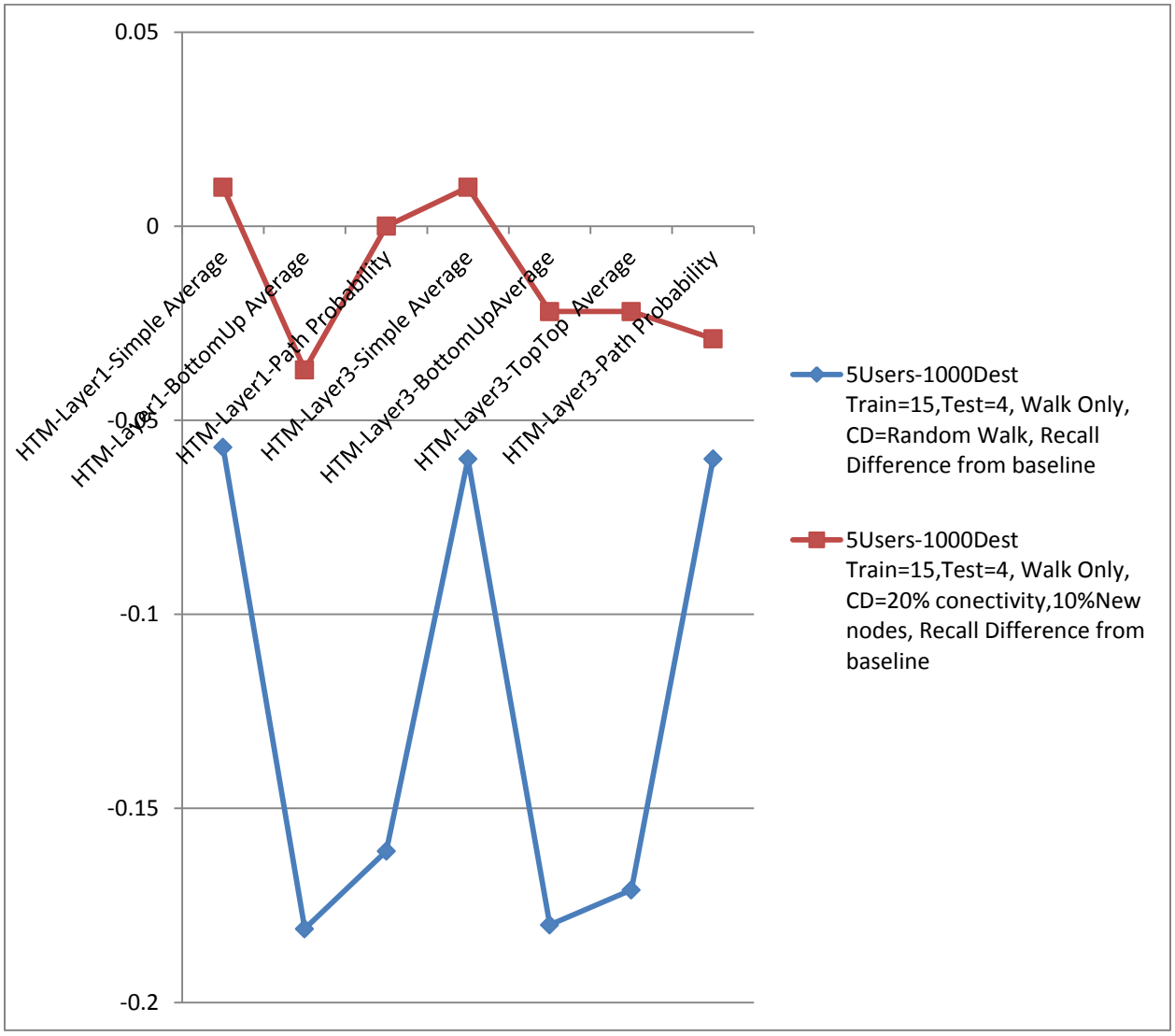


Figure 103 Appendix B Recall difference between baseline and Concept Drift for 15 Train, 4 Test, 1000 Destinations

Twenty train days, five test day, one thousand destinations and five users

The graph below supports the recall statistics for 20 train days, 5 test days and 1000 destinations.

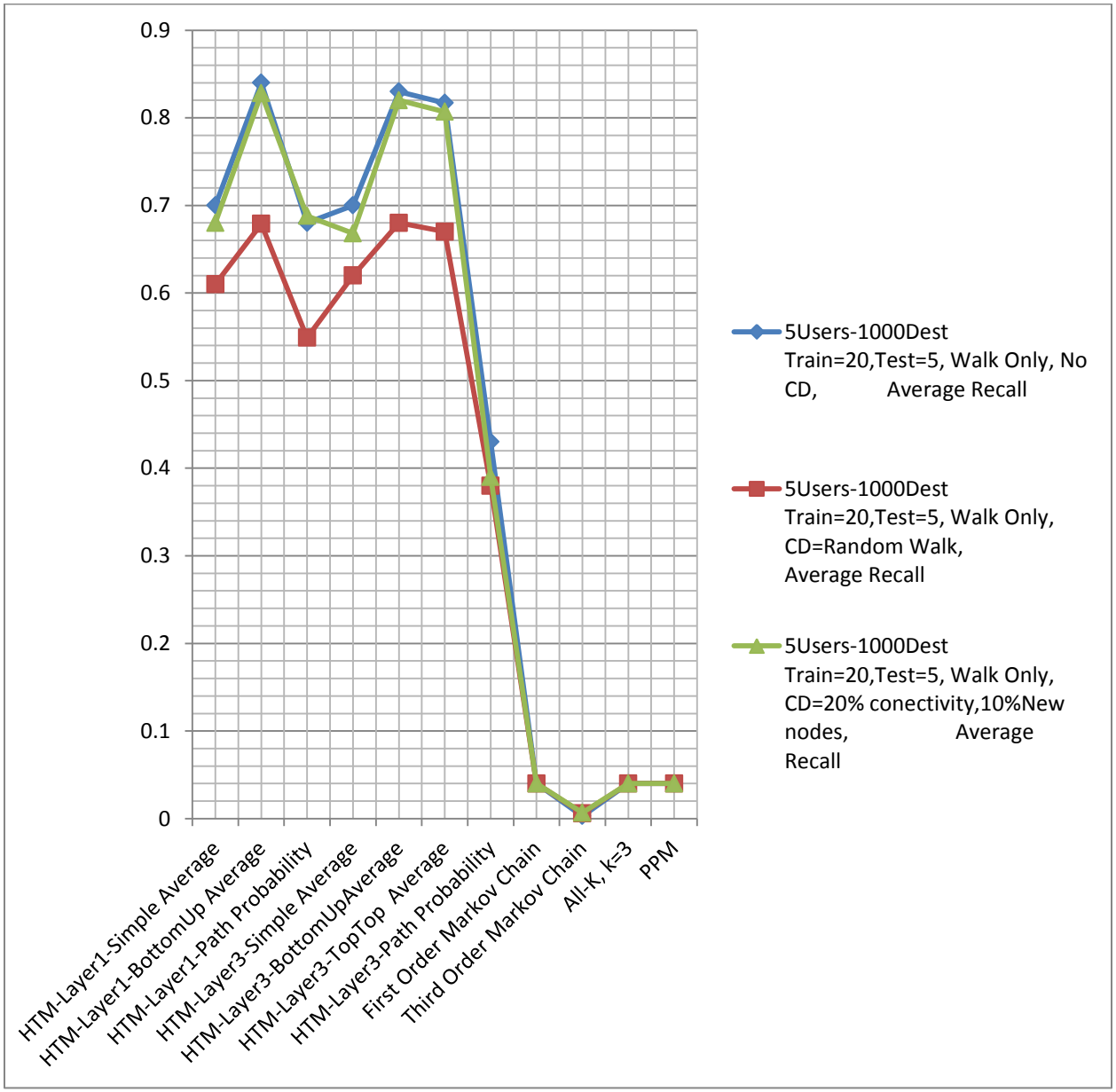


Figure 104 Appendix B Synthetic Data Recall Results with Concept Drift 20 Train, 5 Test, 1000 Destinations

The graph below supports the precision statistics for 20 train days, 5 test days and 1000 destinations.

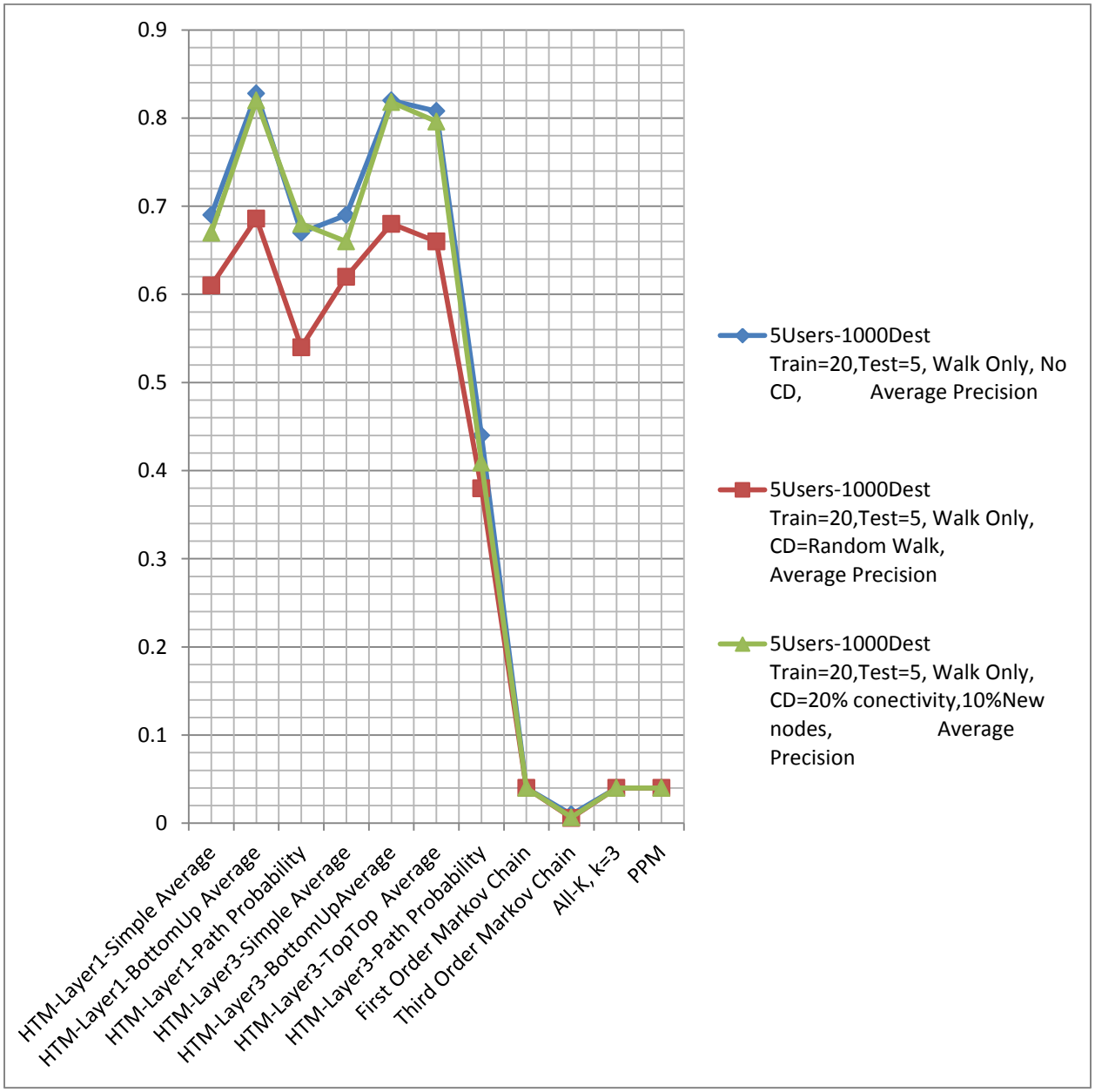


Figure 105 Appendix B Synthetic Data Precision Results with Concept Drift 20 Train, 5 Test, 1000 Destinations

The next graphs report false negative and false positive statistics for 20 train days, 5 test days, and 1000 destinations.

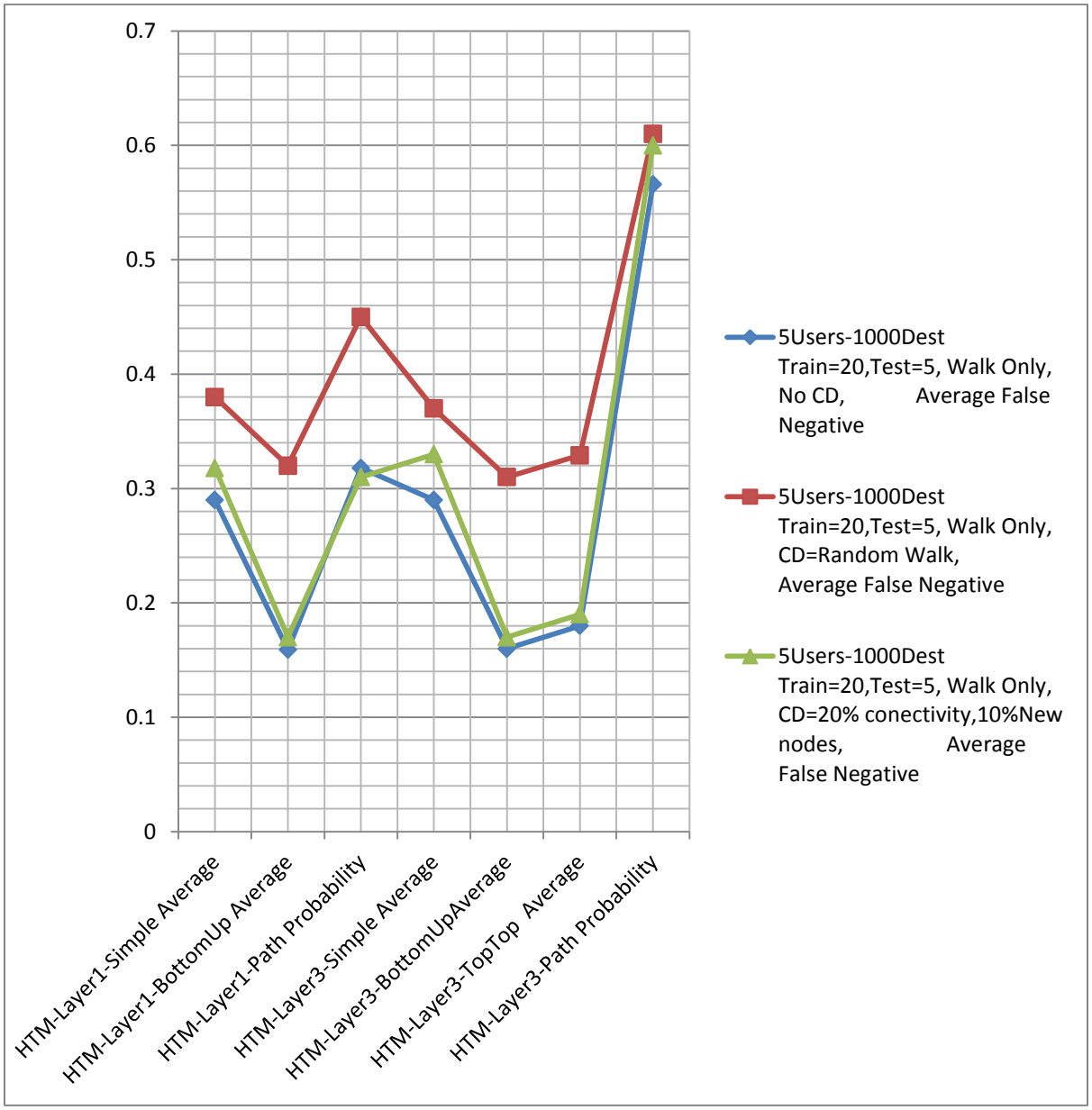


Figure 106 Appendix B Synthetic Data False Negatives Results with Concept Drift 20 Train, 5 Test, 1000 Destinations

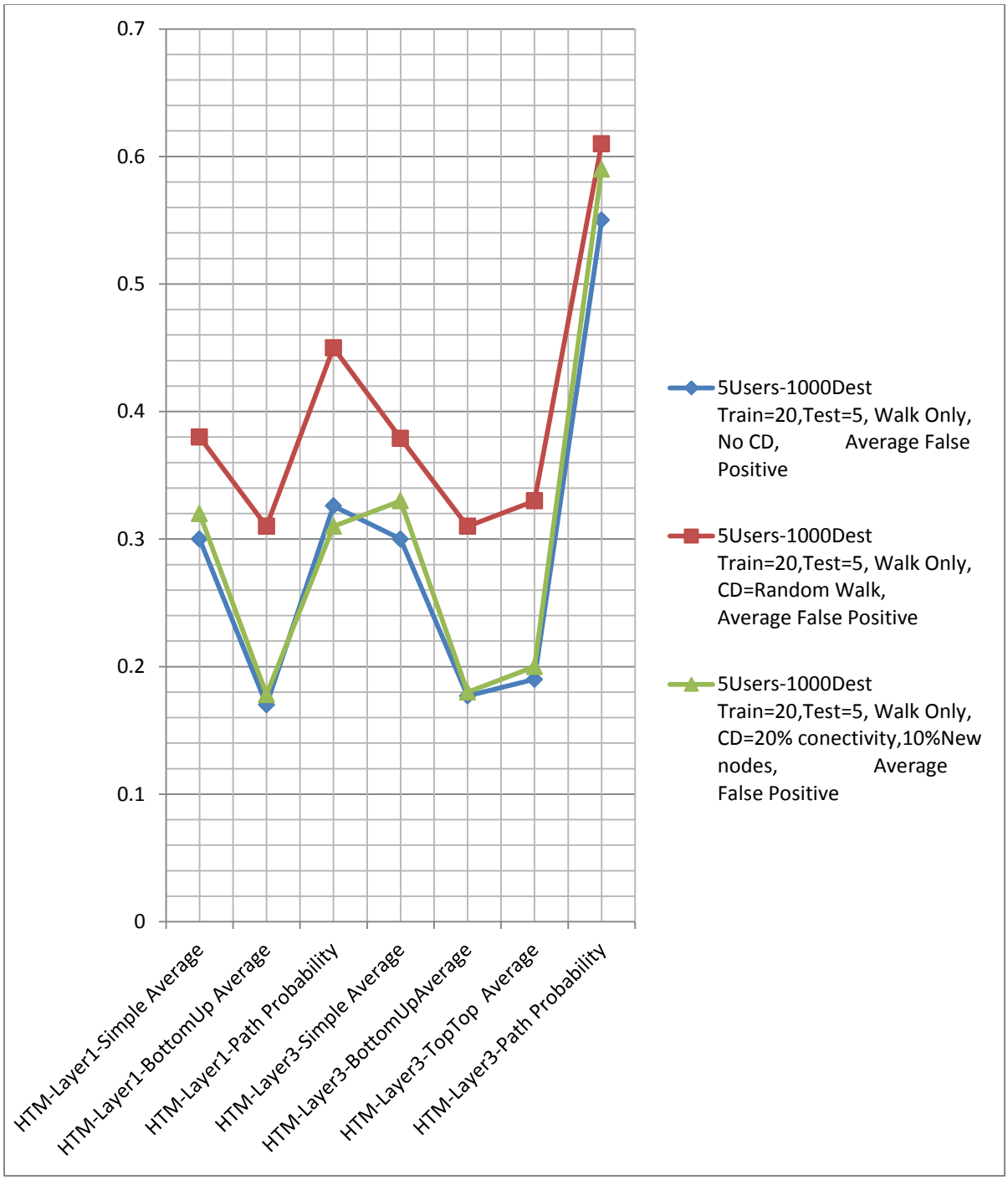


Figure 107 Appendix B Synthetic Data False Positives Results with Concept Drift 20 Train, 5 Test, 1000 Destinations

Recall difference between the baseline (at zero) and the random walk and context drift algorithms.

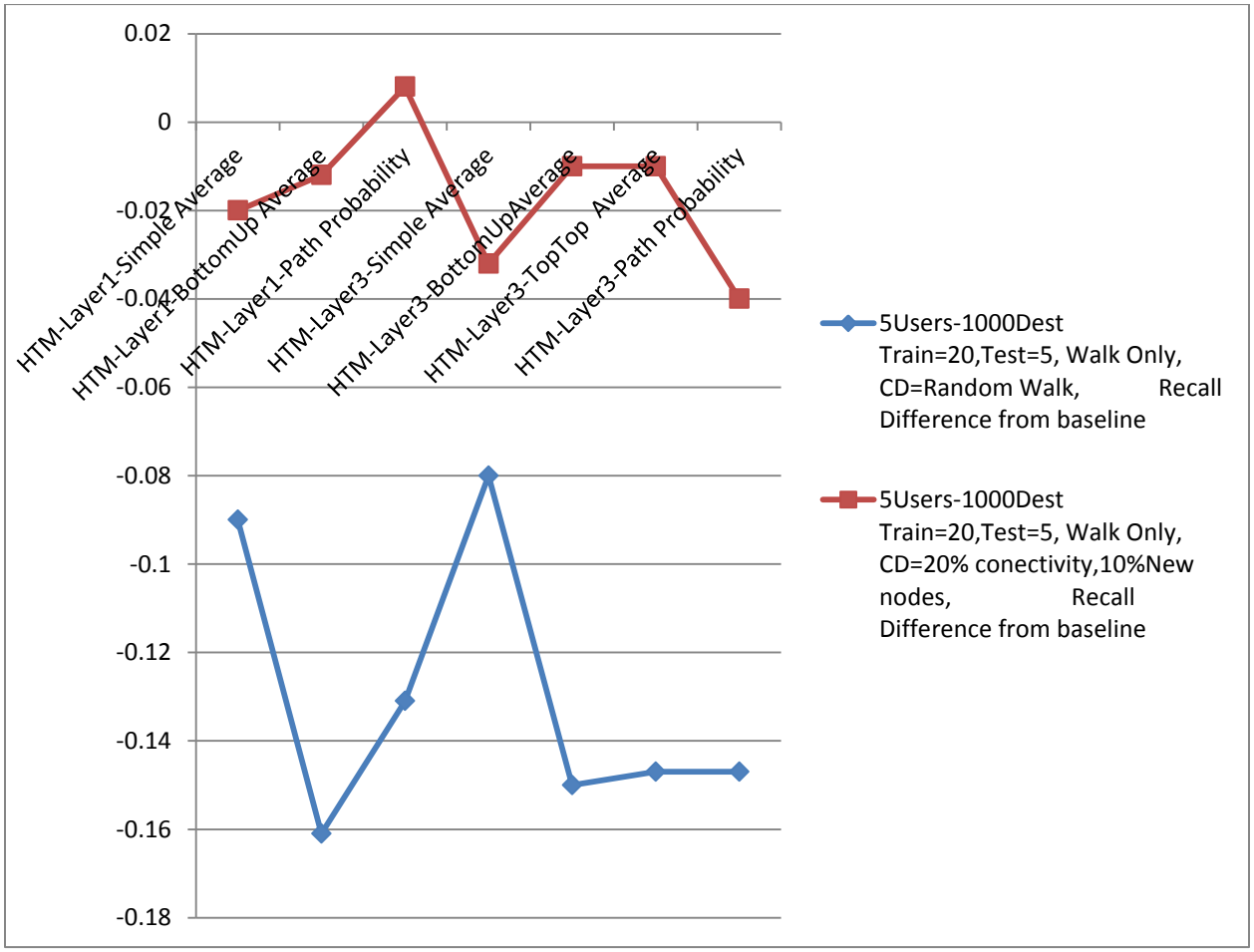


Figure 108 Appendix B Recall difference between baseline and Concept Drift for 20 Train, 5 Test, 1000 Destinations

Appendix C

Intra Observation Repetitiveness MATLAB Algorithm

```
% Computes the result as Intra Observation percentage of
repeated elemenst over size of observation.
% Note that a result of 75% over an observation of size 4 means
that 75% of
 elements in the observation repeat, that is 3 in 4, as in input
= [3 3 3 3]
input = data;
observation_size = length(input);
range = length(input(:,1));
result = [];
for i=1:range
    unique_over_input =
numel(unique(input(i,:)))/observation_size;
    repeated = 1 - unique_over_input;
    result = [result repeated];
end
mean(result)
```

Appendix D

Inter Observation Repetitiveness MATLAB Algorithm

```
input = data;
range = length(input(:,1));
result = [];
for i=1:range
    mode_val = mode(input(i,:));
    if length(unique(input(i,:))) ==  length(input(i,:))
        out = 'At least one observation is completely unique'
    end
    result = [result mode_val];
end

 unique_across_input = numel(unique(result))/length(result);
 repeated = 1 - unique_across_input;
 ;result
 ;hist(result,100);figure(gcf);
 repeated
```

Appendix E

User Attribution Test Results when simulating DOS attacks

Experiments using synthetic data for ten users and four infected users

The next graph presents before and after recall and precision statistics for DOS attacks against 5 destinations sent within 5 milliseconds spaced by 5 milliseconds.

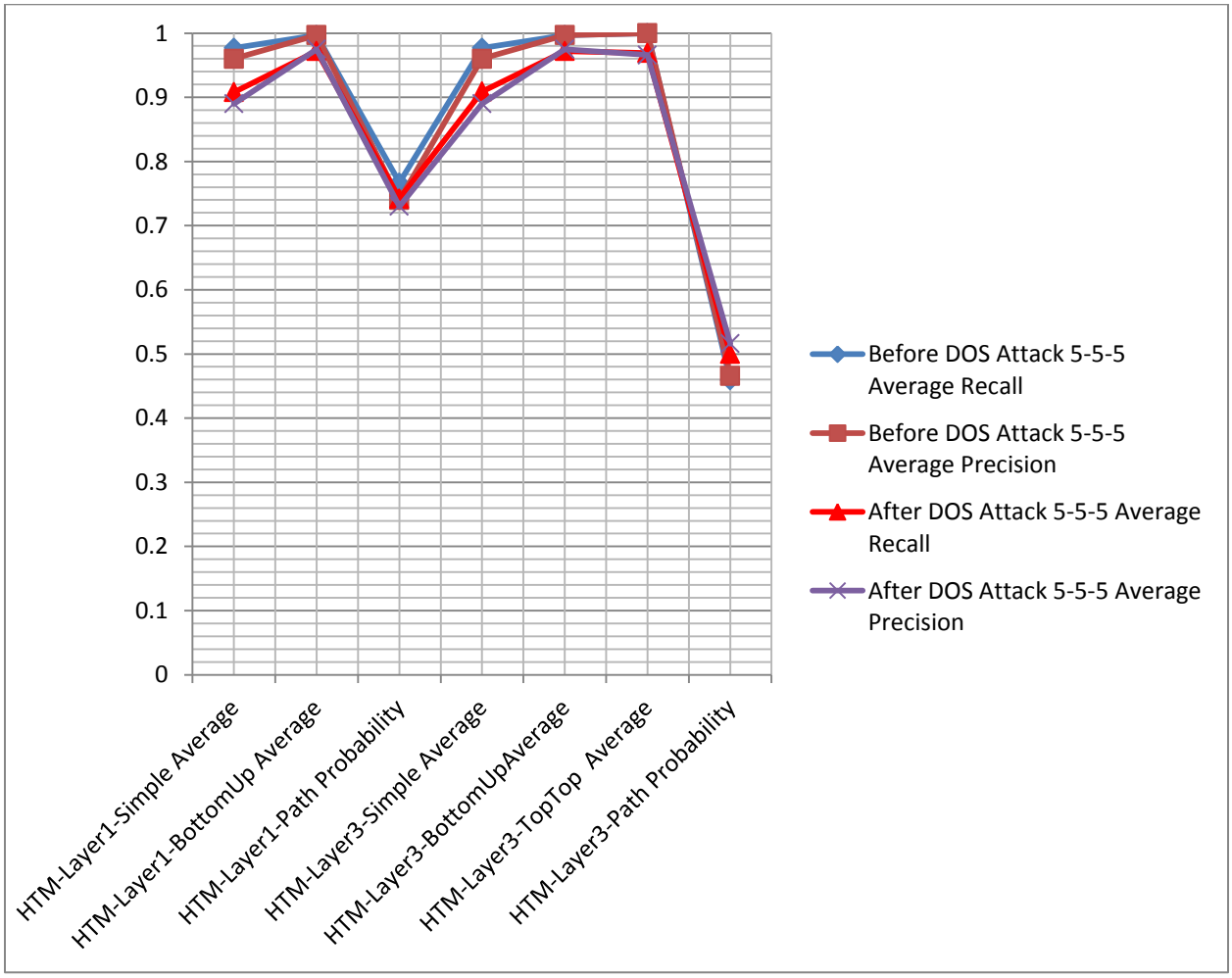


Figure 109 Appendix E 5-5-5 DOS Attack Recall & Precision results before and after Attack using Synthetic Data

The next graph presents before and after recall and precision statistics for DOS attacks against 10 destinations sent within 10 milliseconds spaced by 5 milliseconds.

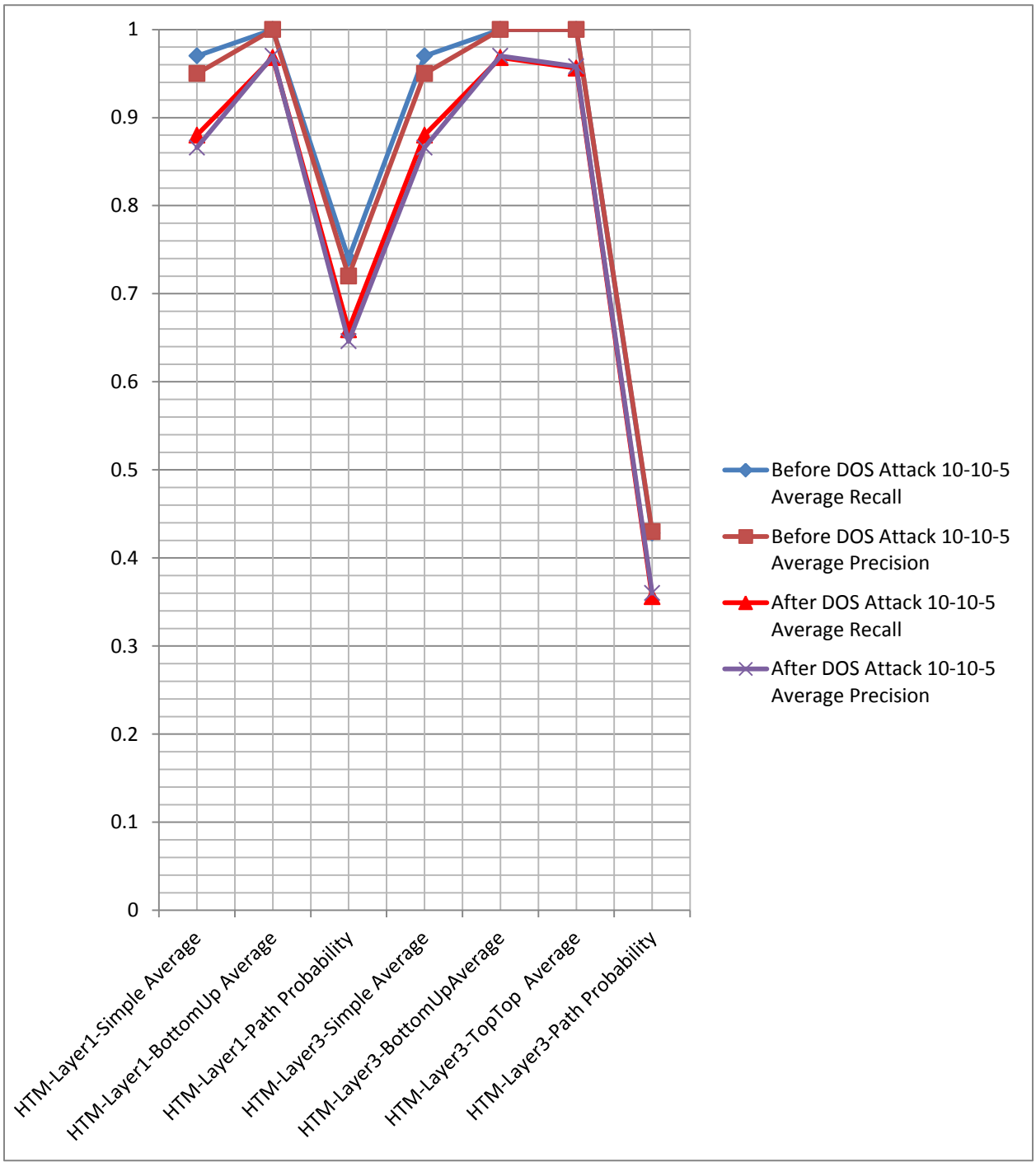


Figure 110 Appendix E 10-10-5 DOS Attack Recall & Precision results before and after Attack using Synthetic Data

The next graph presents before and after recall and precision statistics for DOS attacks against 20 destinations sent within 20 milliseconds spaced by 5 milliseconds.

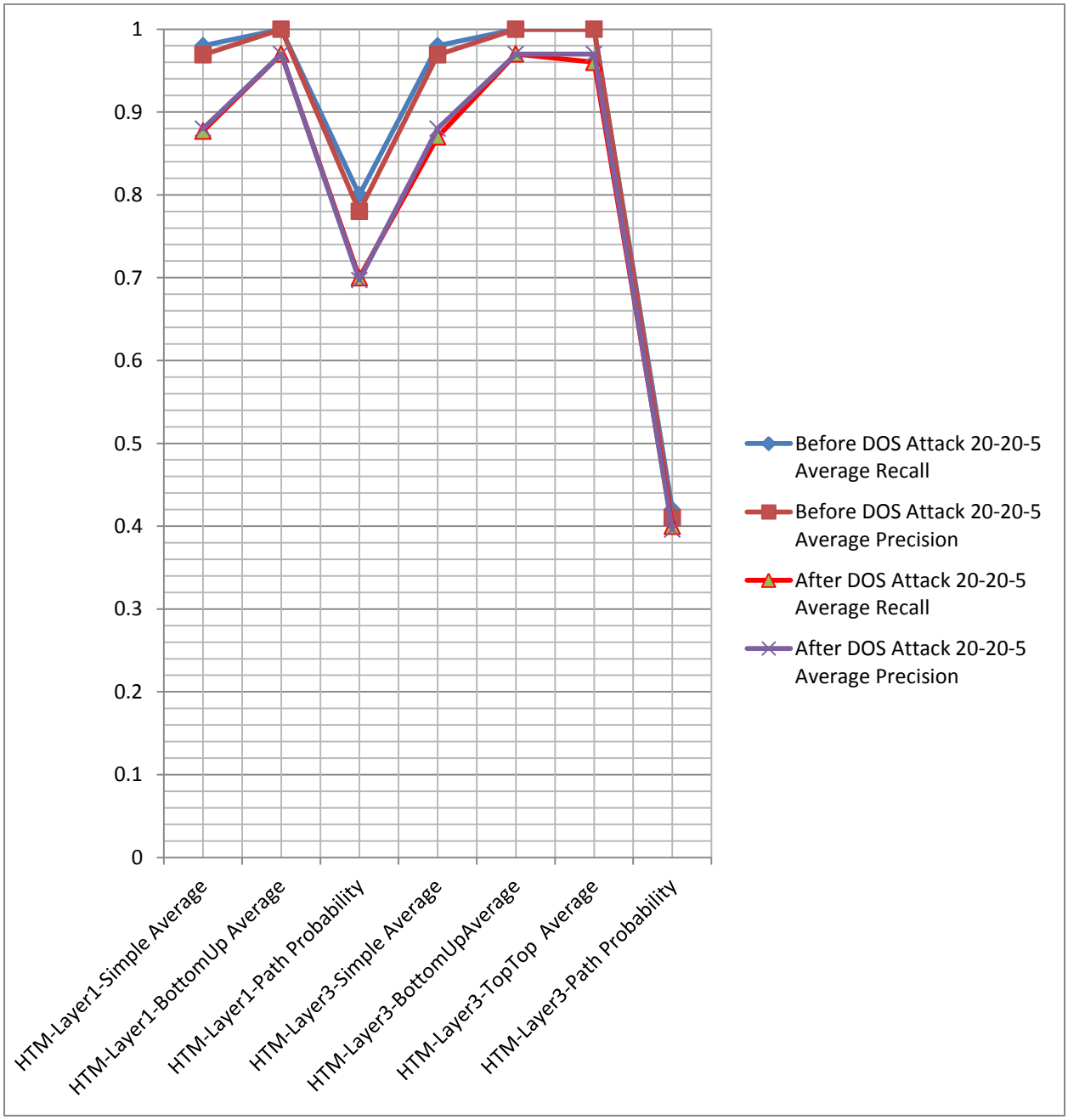


Figure 111 Appendix E 20-20-5 DOS Attack Recall & Precision results before and after Attack using Synthetic Data

Experiments using real network data for ten users and four infected users

The next graph presents before and after recall and precision statistics for DOS attacks against 5 destinations sent within 5 milliseconds spaced by 5 milliseconds.

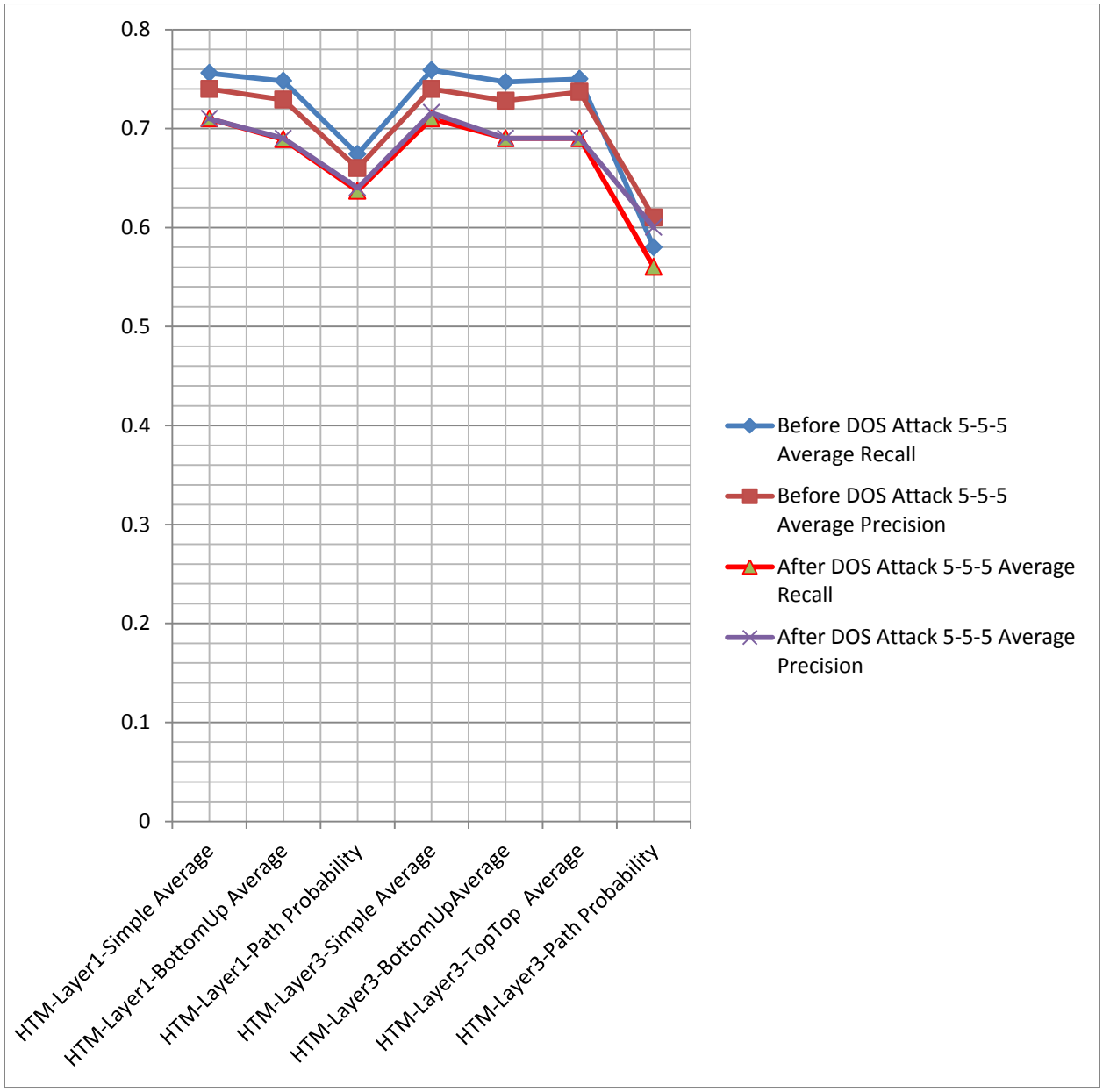


Figure 112 Appendix E 5-5-5 DOS Attack Recall & Precision results before and after Attack using Real Data

The next graph presents before and after recall and precision statistics for DOS attacks against 10 destinations sent within 10 milliseconds spaced by 5 milliseconds.

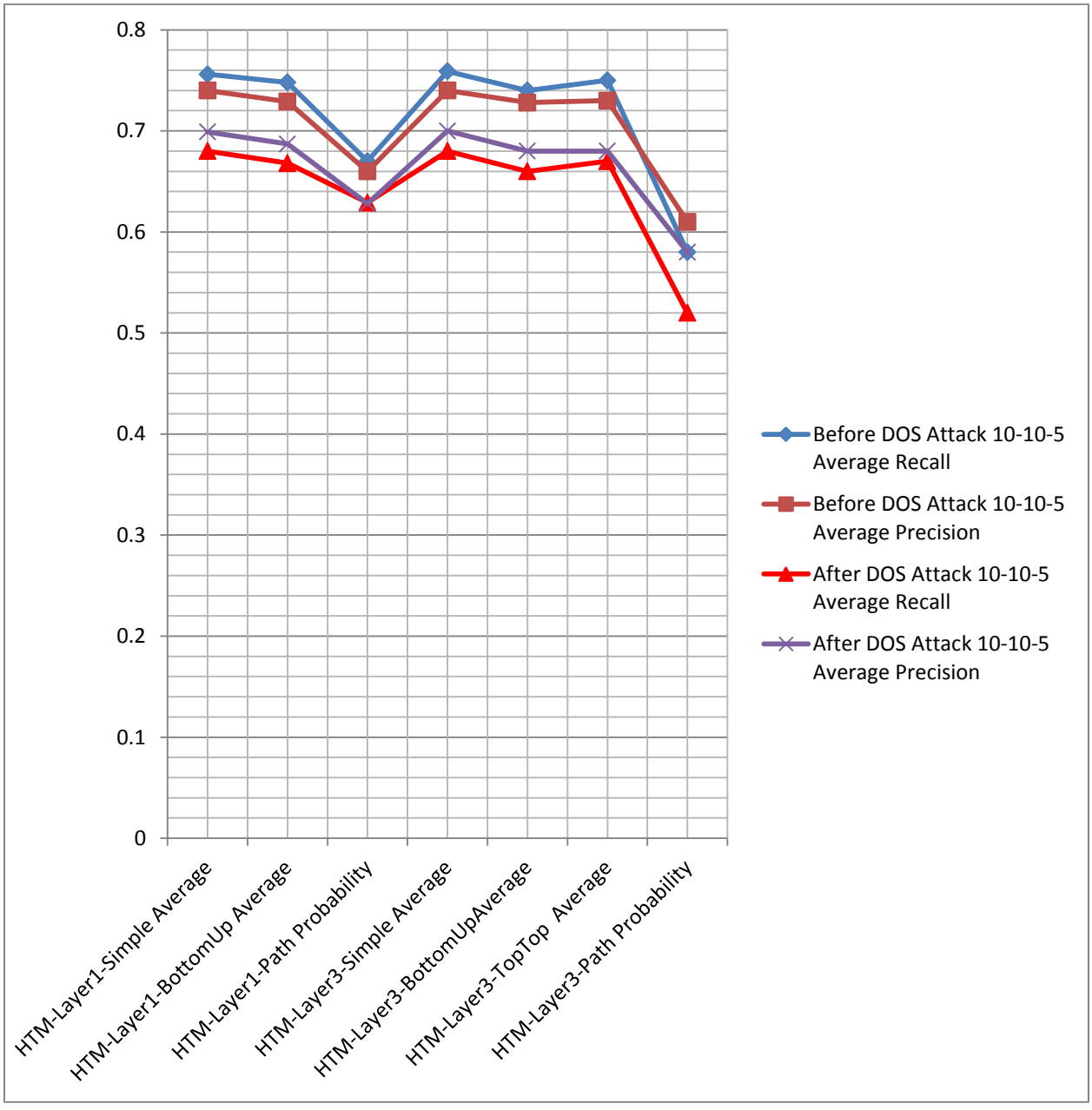


Figure 113 Appendix E 10-10-5 DOS Attack Recall & Precision results before and after Attack using Real Data

The next graph presents before and after recall and precision statistics for DOS attacks against 20 destinations sent within 20 milliseconds spaced by 5 milliseconds.

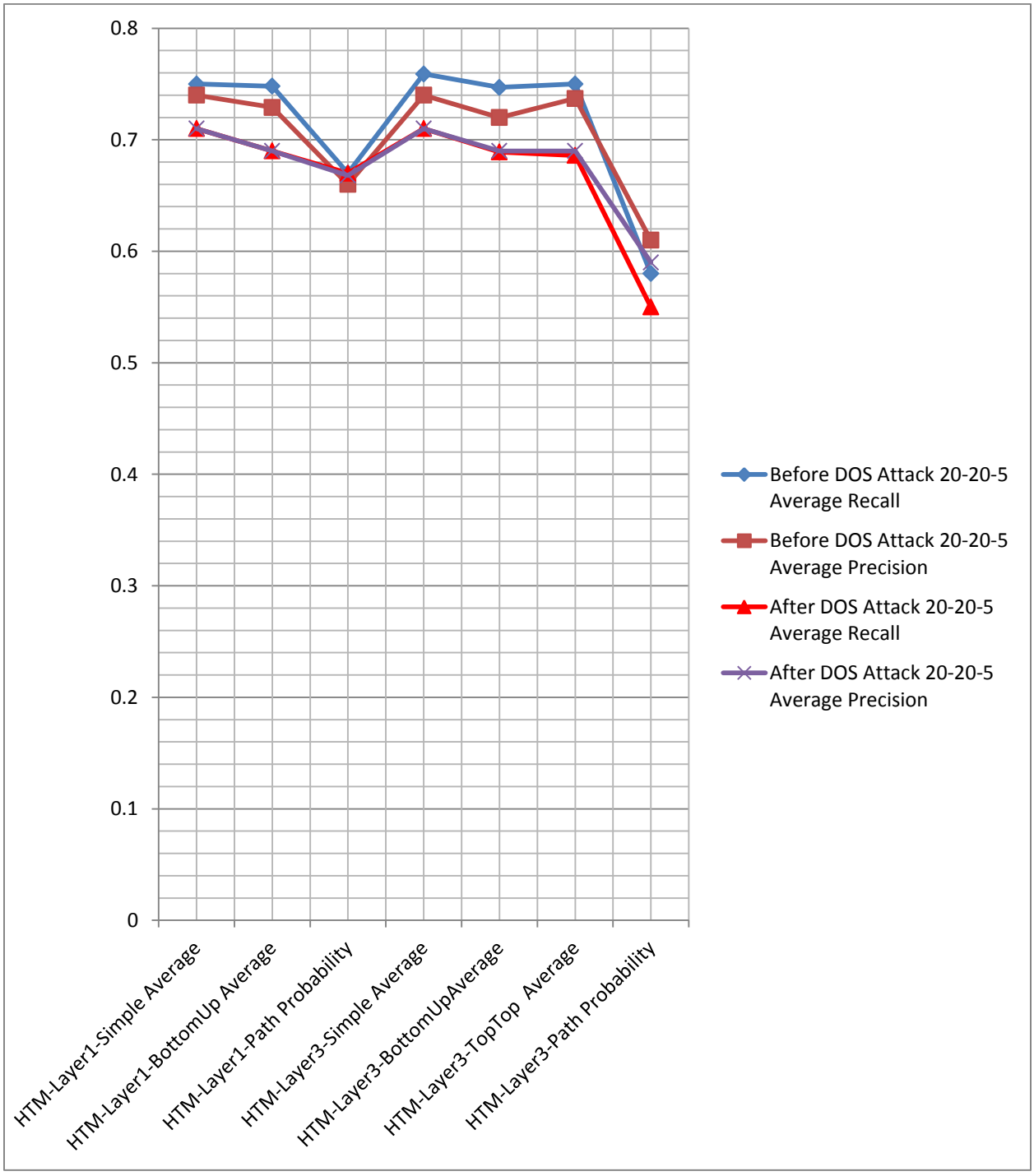


Figure 114 Appendix E 20-20-5 DOS Attack Recall & Precision results before and after Attack using Real Data

Appendix F

User Attribution Test Results when simulating Phish attacks

Experiments using synthetic data for ten users and four infected users

The next graph presents before and after recall and precision statistics for Phish attacks against 1 destination sent within 1 millisecond spaced by a random uniform time between 1 millisecond and 1 hour.

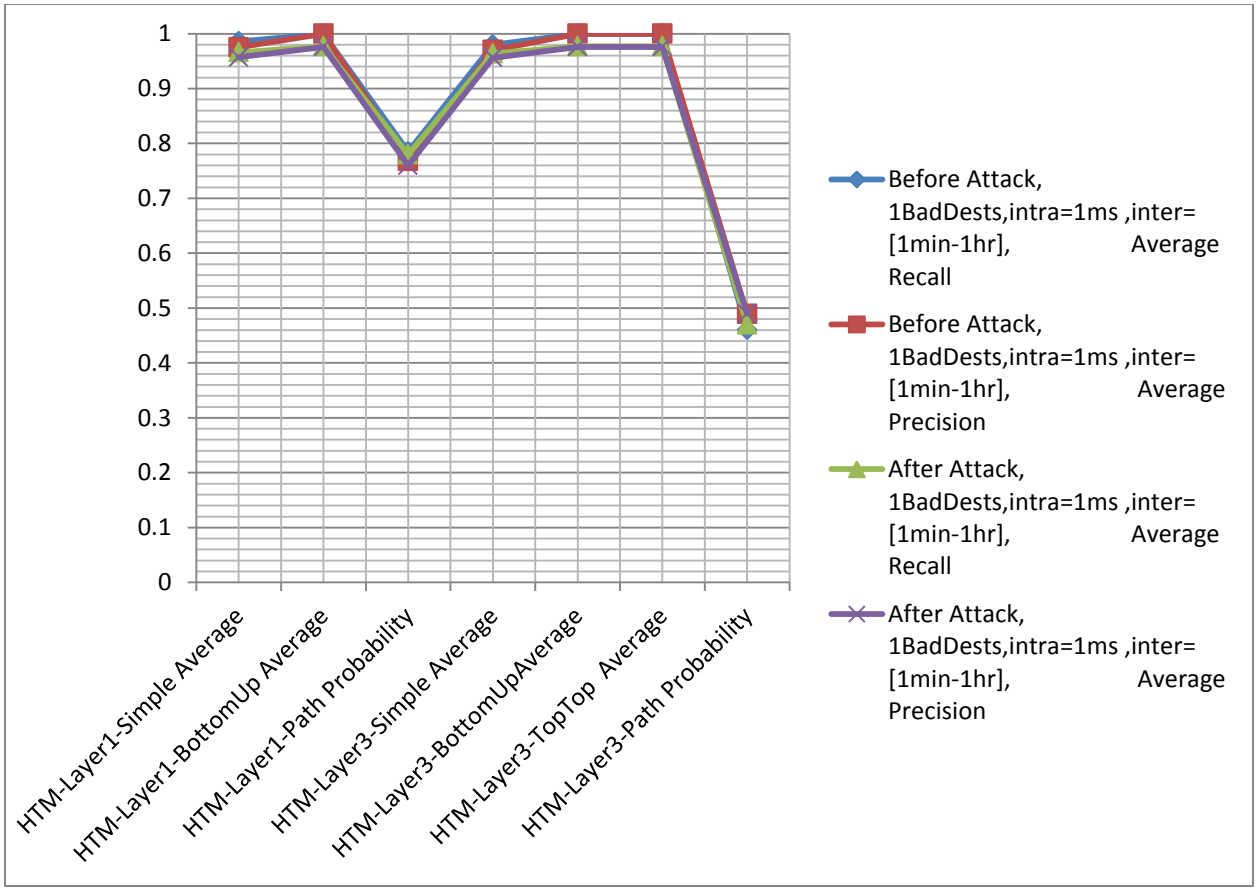


Figure 115 Appendix F 1-1-1 Phish Attacks Recall & Precision results before and after Attack using Synthetic Data

The next graph presents before and after recall and precision statistics for Phish attacks against 3 destinations sent within 3 milliseconds spaced by a random uniform time between 1 millisecond and 1 hour.

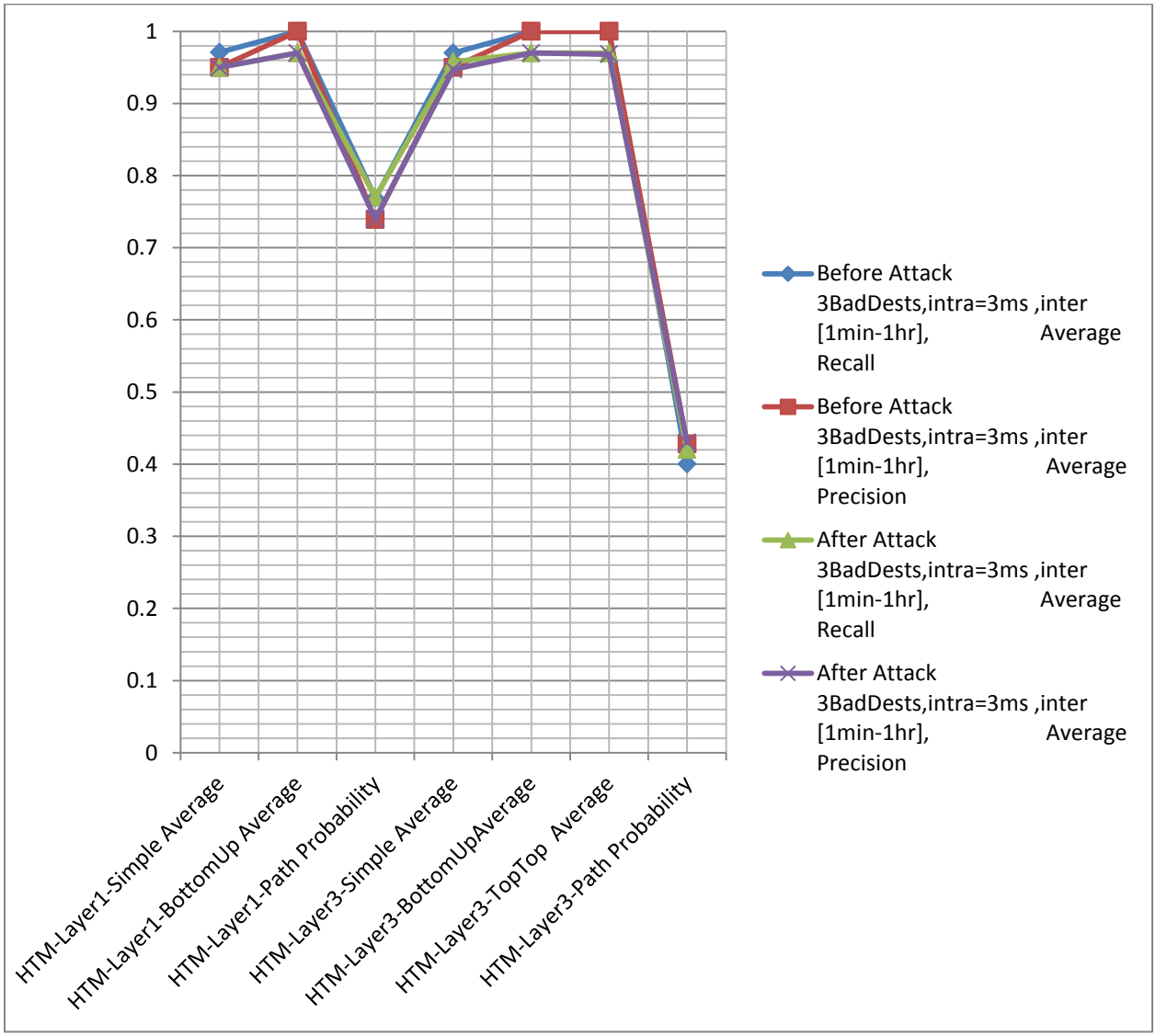


Figure 116 Appendix F 3-3-1 Phish Attacks Recall & Precision results before and after Attack using Synthetic Data

The next graph presents before and after recall and precision statistics for Phish attacks against 5 destinations sent within 5 milliseconds spaced by a random uniform time between 1 millisecond and 1 hour.

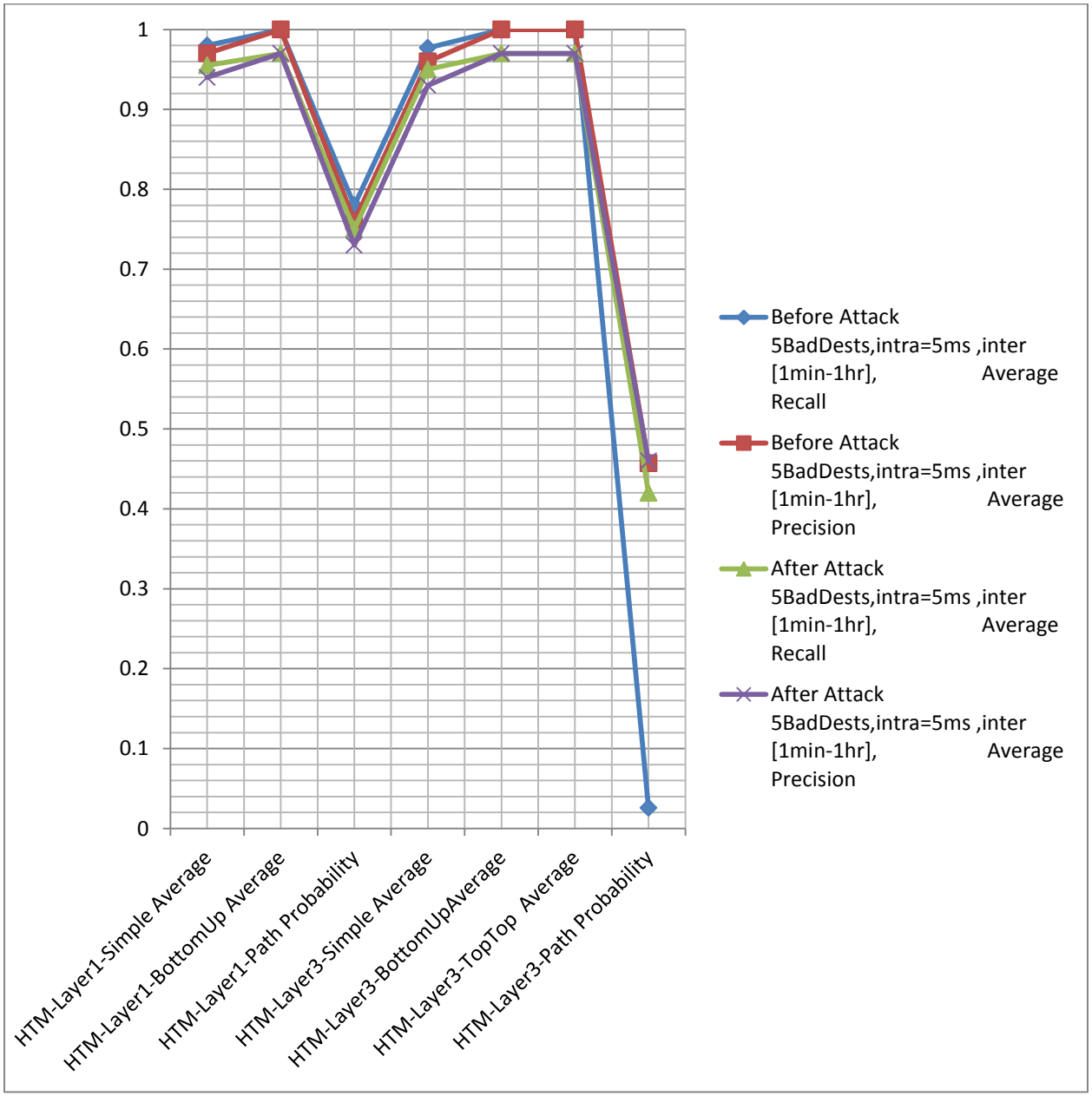


Figure 117 Appendix F 5-5-1 Phish Attacks Recall & Precision results before and after Attack using Synthetic Data

Experiments using real network data for ten users and four infected users

The next graph presents before and after recall and precision statistics for Phish attacks against 1 destination sent within 1 millisecond spaced by a random uniform time between 1 millisecond and 1 hour.

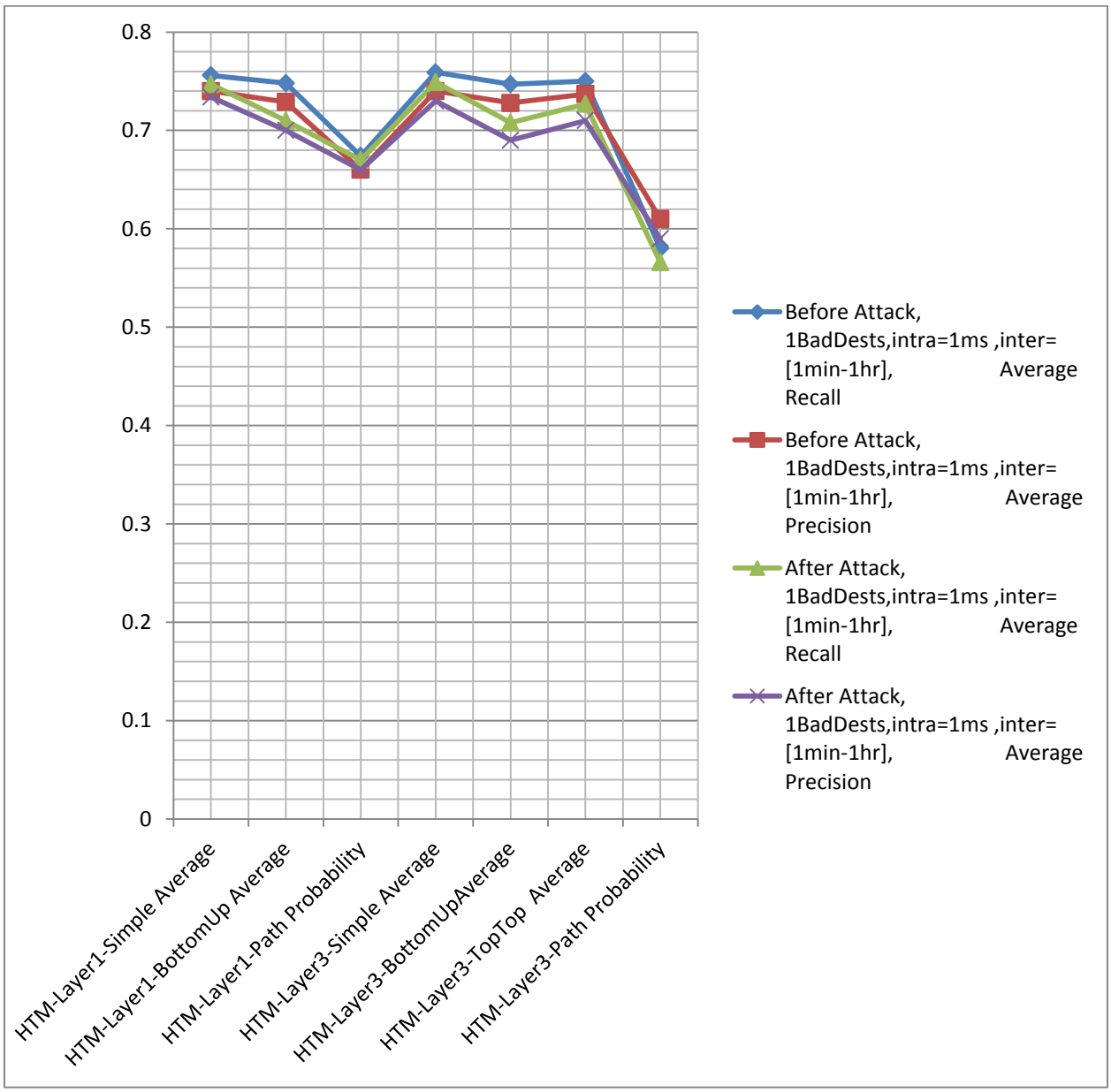


Figure 118 Appendix F 1-1-1 Phish Attacks Recall & Precision results before and after Attack using Real Data

The next graph presents before and after recall and precision statistics for Phish attacks against 3 destinations sent within 3 milliseconds spaced by a random uniform time between 1 millisecond and 1 hour.

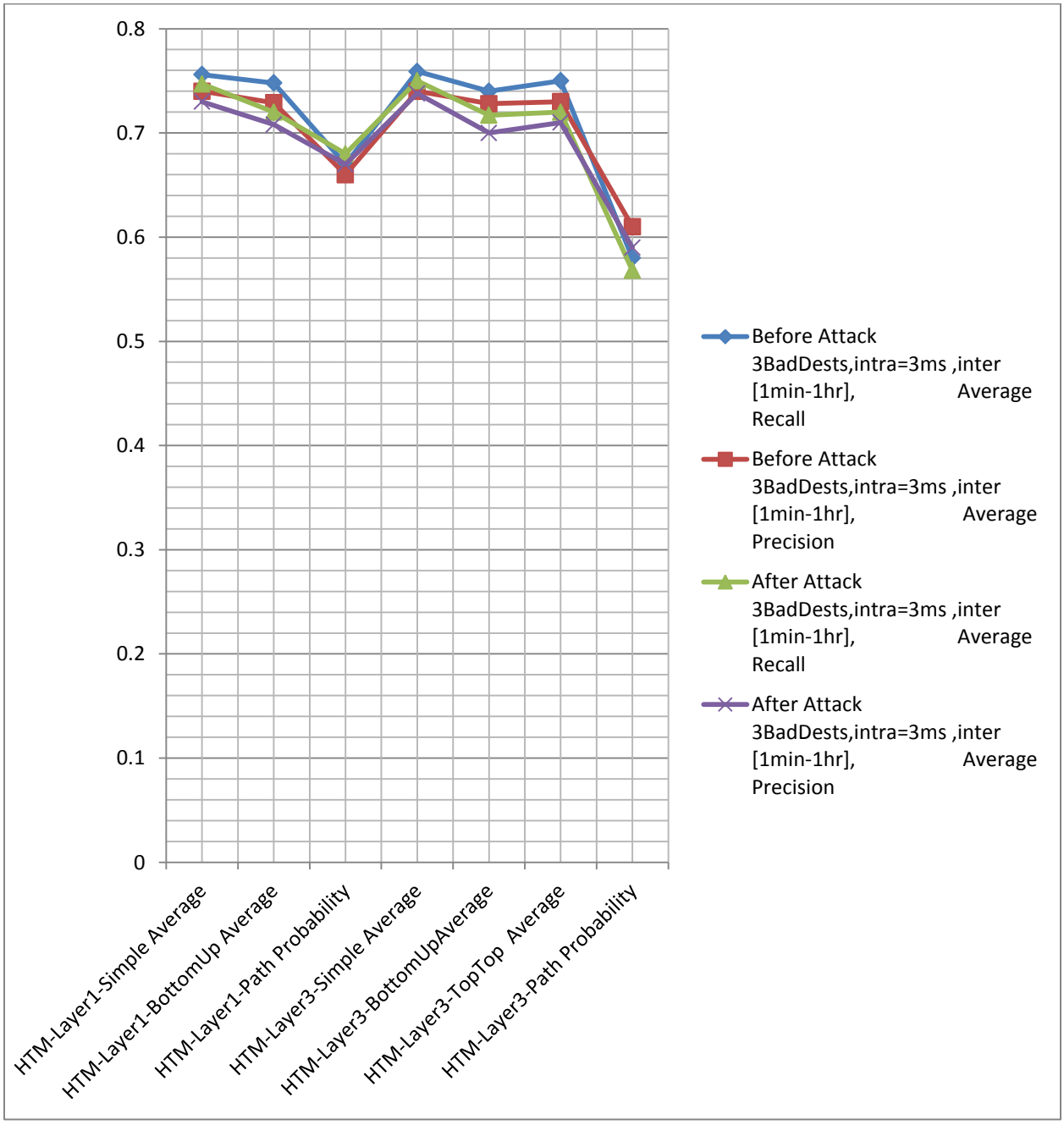


Figure 119 Appendix F 3-3-1 Phish Attacks Recall & Precision results before and after Attack using Real Data

The next graph presents before and after recall and precision statistics for Phish attacks against 5 destinations sent within 5 milliseconds spaced by a random uniform time between 1 millisecond and 1 hour.

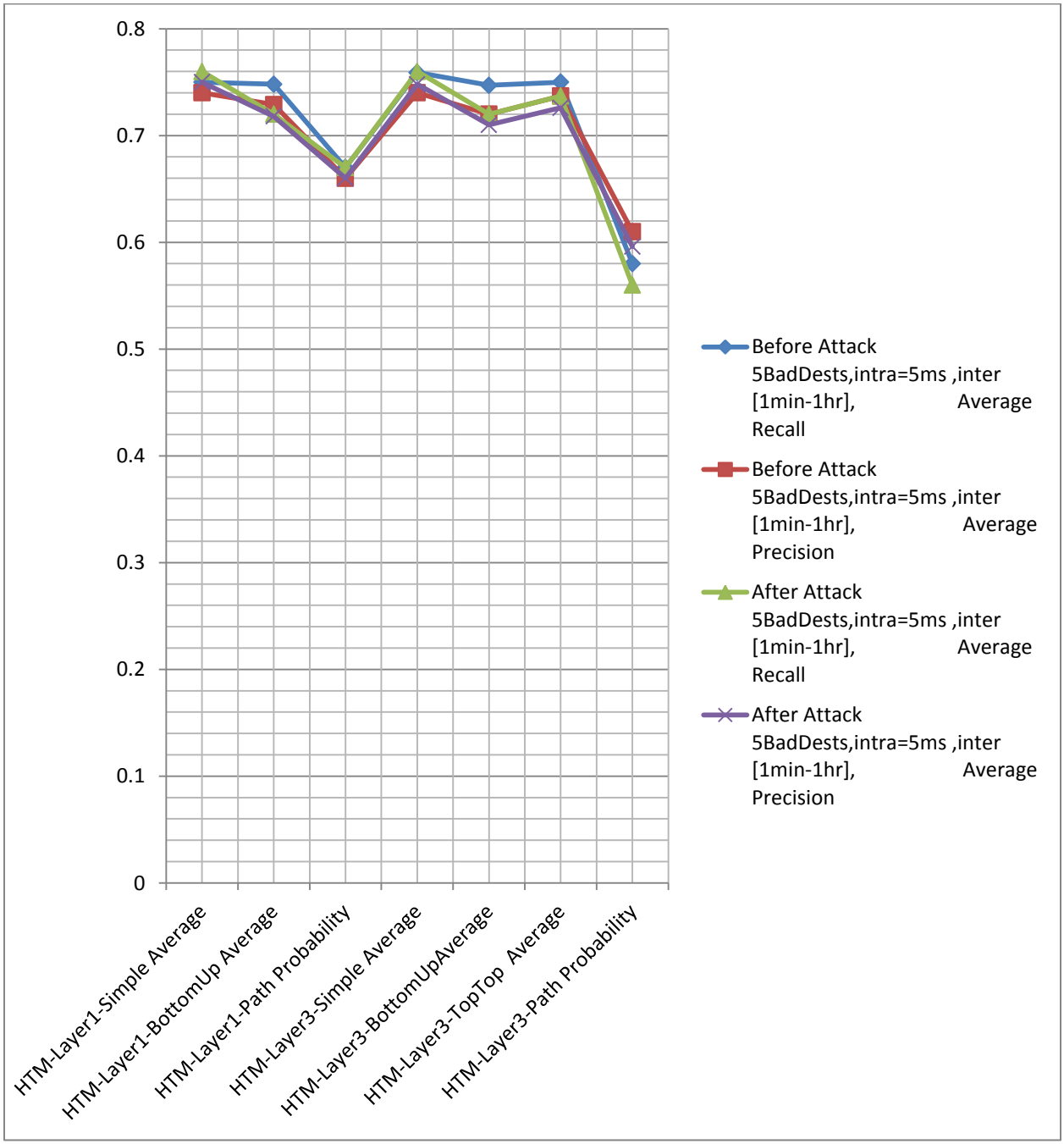


Figure 120 Appendix F 5-5-1 Phish Attacks Recall & Precision results before and after Attack using Real Data

Appendix G

HTM2 (++) User Attribution Test Results Using Real Data

Five Users for five train days, one test day and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (5 train days, 1 test day, 5000 destinations) run against HTM2.

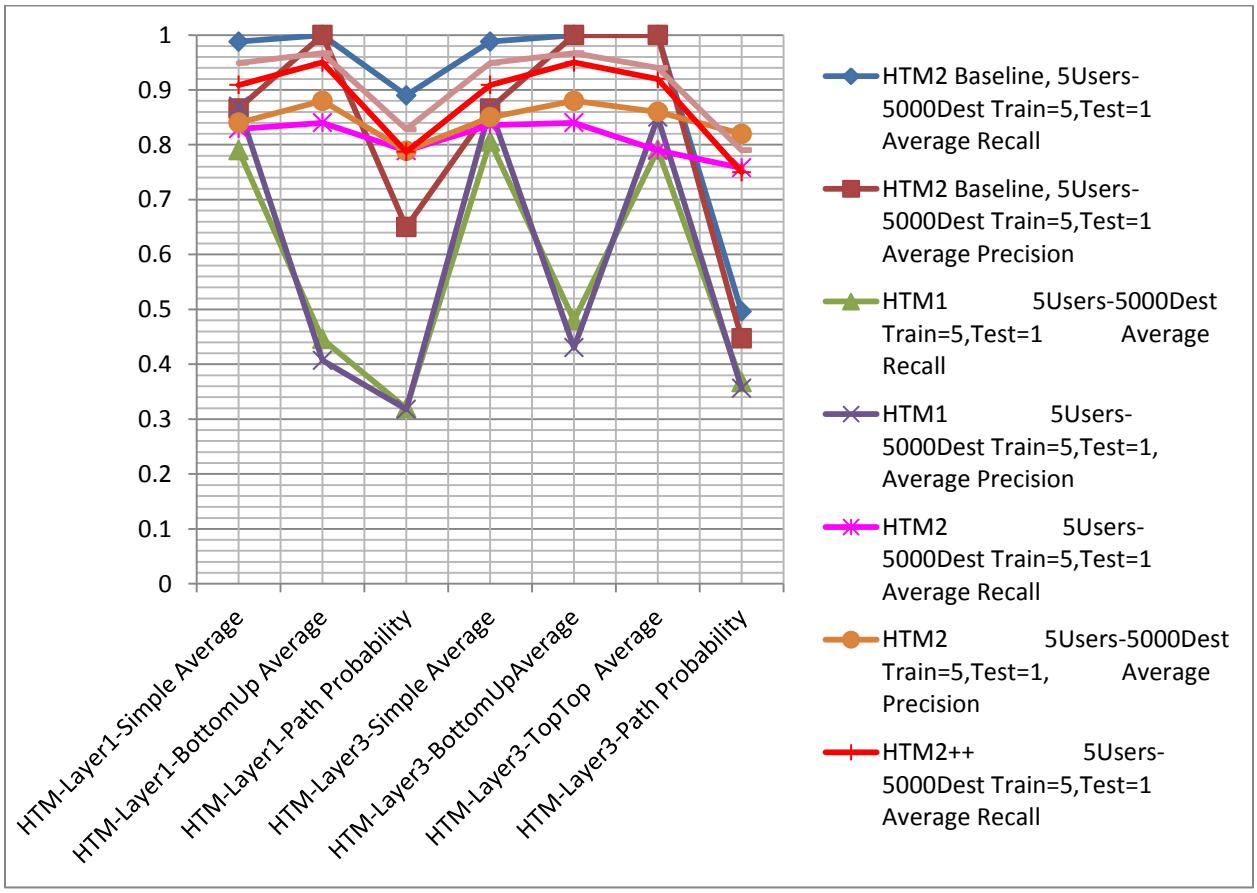


Figure 121 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 5 users, 5 Train, 1 Test

Five users for five train days, two test days and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (5 train days, 2 test days, 5000 destinations) run against HTM2.

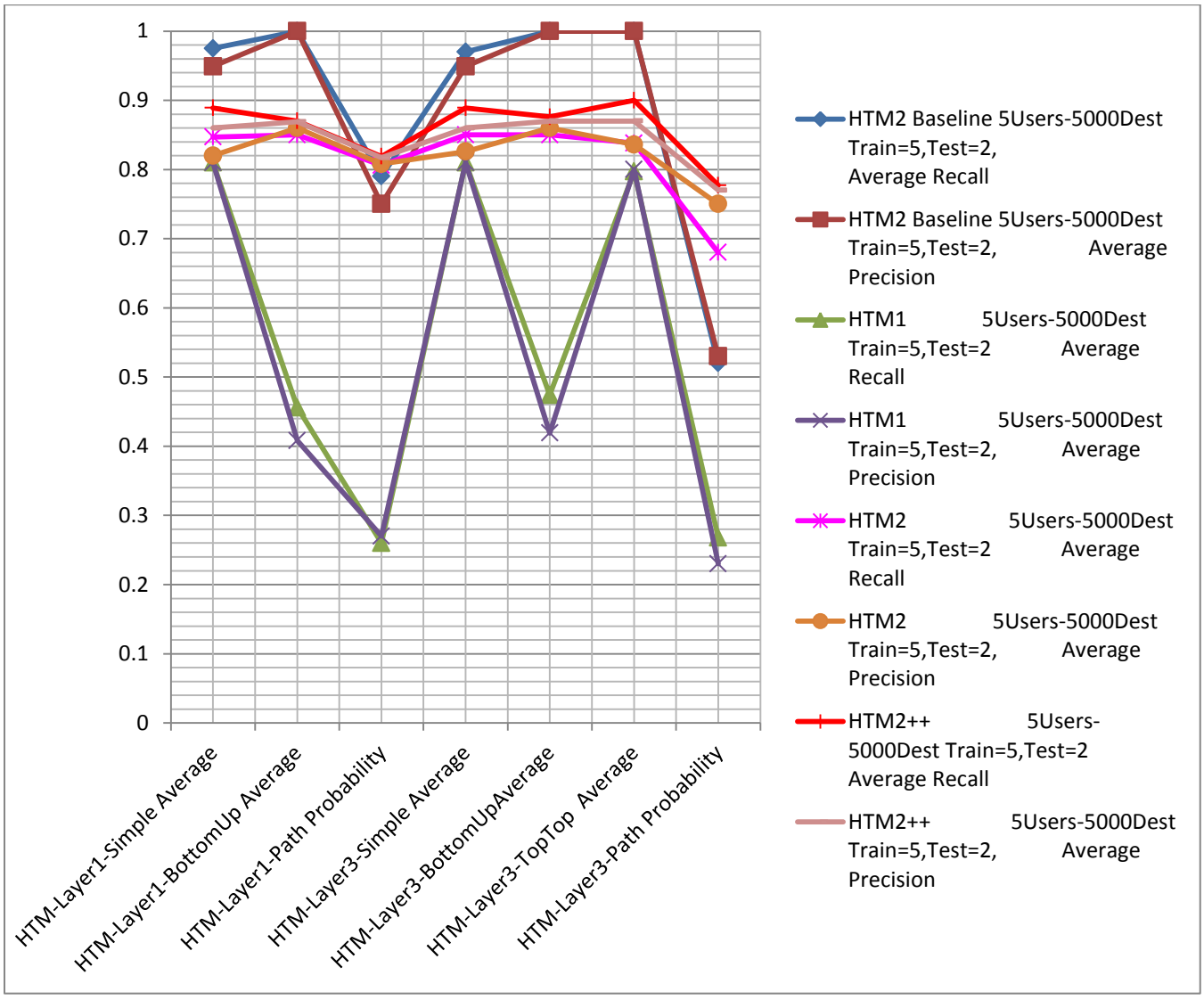


Figure 122 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 5 users, 5 Train, 2 Test

Five users for ten train days, three test days and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (10 train days, 3 test days, 5000 destinations) run against HTM2.

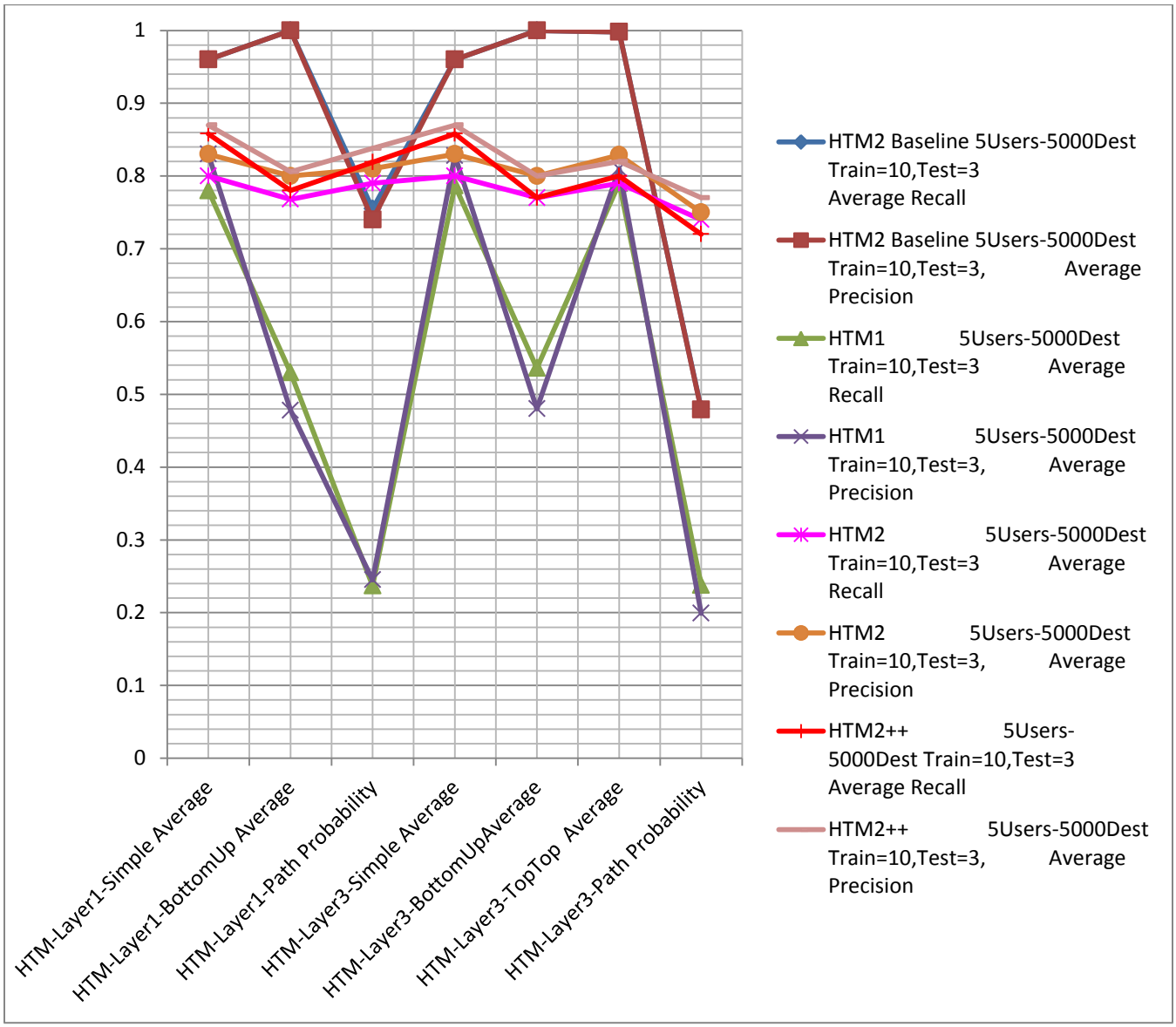


Figure 123 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 5 users, 5 Train, 3 Test

Ten Users for five train days, one test day and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (5 train days, 1 test day, 5000 destinations) run against HTM2.

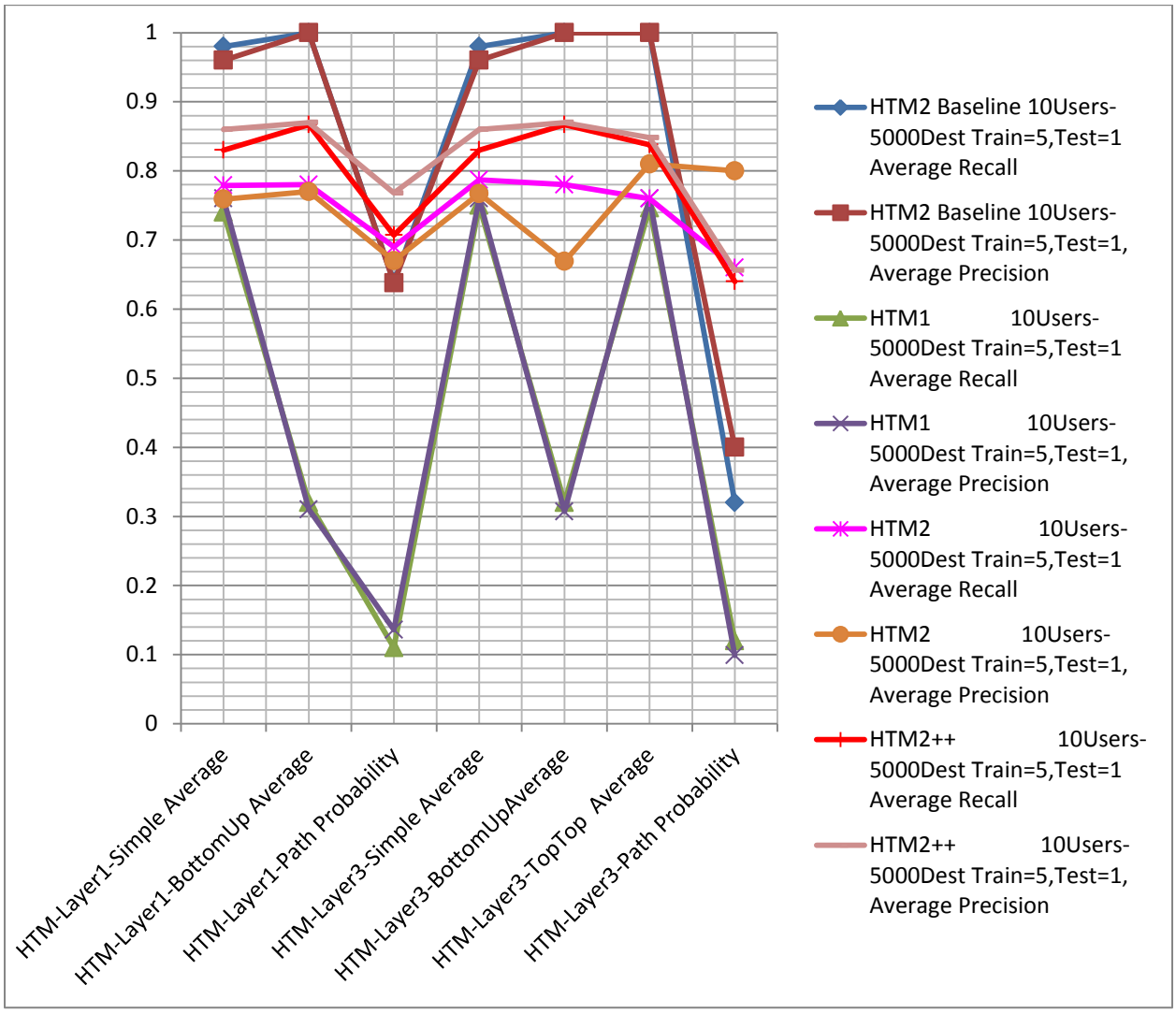


Figure 124 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 10 users, 5 Train, 1 Test

Ten users for five train days, two test days and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (5 train days, 2 test days, 5000 destinations) run against HTM2.

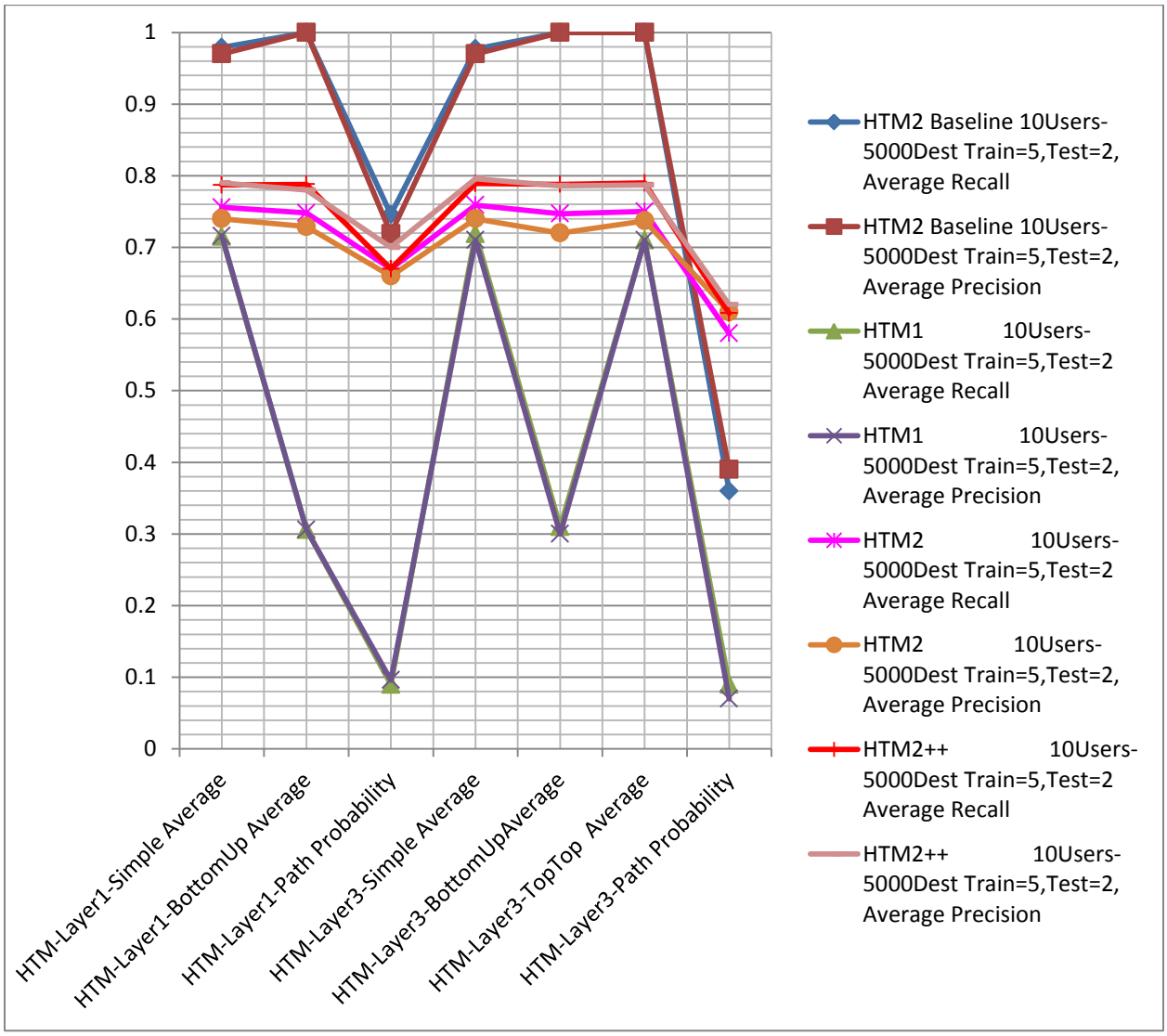


Figure 125 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 10 users, 5 Train, 2 Test

Ten users for ten train days, three test days and over five thousand destinations

The graphs below compare recall and precision statistics for HTM1, HTM2 and HTM2++ (HTM2 run on real network data where same time destinations are removed) run against real network data against a baseline of synthetic data (10 train days, 3 test days, 5000 destinations) run against HTM2.

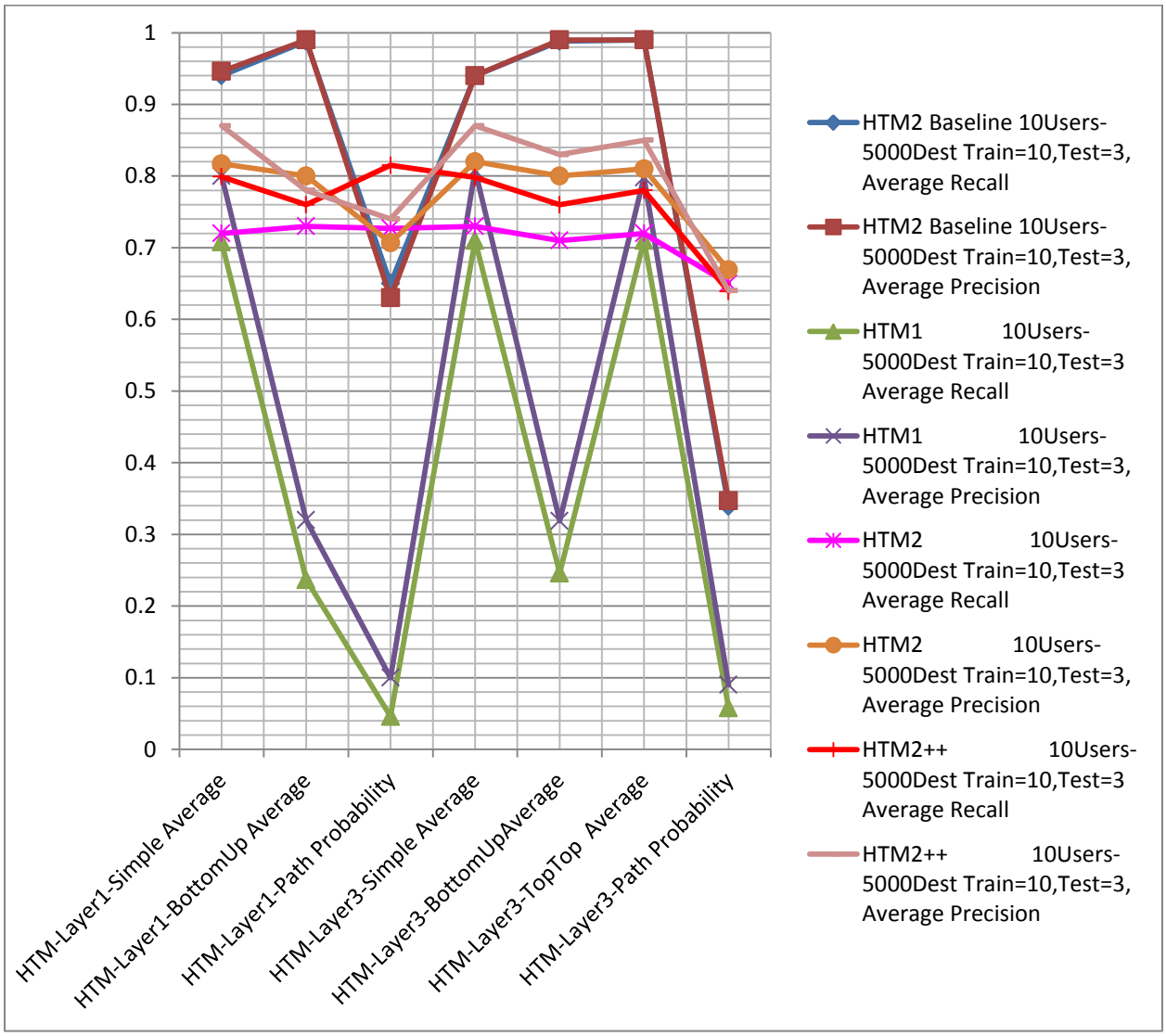


Figure 126 Appendix G HTM1, HTM2, HTM2++ Real Network Data Recall & Precision Results for 10 users, 10 Train, 3 Test

Appendix H

Calibration Results for qualification of HTM V1, HTM V2 and Alternate Algorithms

Calibration runs for qualifying HTM V1 with Synthetic Data

Users	Layer 1 HTM Algorithms % Recall Accuracy	Layer 3 HTM Algorithm % Recall Accuracy
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
		Average Top-Top
User-1		100%
User-2		100%
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%

Table 21 Appendix H – Calibrations HTMV1 with Synthetic Data

Calibration runs which failed to qualify HTM V1 with Real Network Data

Users	Layer 1 HTM Algorithms % Recall Accuracy	Layer 3 HTM Algorithm % Recall Accuracy
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	99% (1 error)	99% (1 error)
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	78% (48 errors)	79% (46 errors)
User-2	82% (53 errors)	85% (44 errors)
User-3	81% (22 errors)	81% (21 errors)
User-4	74% (38 errors)	76% (35 errors)
User-5	71% (4 errors)	79% (3 errors)
		Average Top-Top
User-1		100%
User-2		99% (1 error)
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	6% (206 errors)	4% (215 errors)
User-2	27% (221 errors)	22% (236 errors)
User-3	61% (44 errors)	57% (48 errors)
User-4	41% (86 errors)	32% (100 errors)
User-5	36% (9 errors)	36% (9 errors)

Table 22 Appendix H – Calibrations HTMV1 with Real Network Data

Calibration runs for qualifying HTM V2 with Synthetic Data

The HTM version 2 was calibrated accounting for continuous repeated destinations at all layers (1-3) of the HTM during the learning phase. During the inference phase continuous processing of the same destination was limited to layer 1 only. This configuration was used for calibration of HTM version 2 and for all experiments conducted with this version of the HTM as this configuration produced the best results.

Users	**Layer 1 HTM Algorithms % Recall Accuracy**	**Layer 3 HTM Algorithm % Recall Accuracy**
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%
		Average Top-Top
User-1		100%
User-2		100%
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	100%	100%
User-2	100%	100%
User-3	100%	100%
User-4	100%	100%
User-5	100%	100%

Table 23 Appendix H – Calibrations HTMV2 with Synthetic Data

Calibration runs for qualifying HTM V2 with Real Network Data

Users	Layer 1 HTM Algorithms % Recall Accuracy	Layer 3 HTM Algorithm % Recall Accuracy
	Simple Average	*Simple Average*
User-1	100%	100%
User-2	99% (1 error)	100%
User-3	100%	99% (1 error)
User-4	100%	100%
User-5	100%	100%
	Average Bottom Up	*Average Bottom Up*
User-1	99% (1 error)	99% (1 error)
User-2	99% (1 error)	99% (1 error)
User-3	98% (2 errors)	98% (2 errors)
User-4	100%	100%
User-5	100%	100%
		Average Top-Top
User-1		100%
User-2		99% (1 error)
User-3		100%
User-4		100%
User-5		100%
	Path Probability	*Path Probability*
User-1	100%	93% (14 errors)
User-2	99% (3 errors)	96% (12 errors)
User-3	100%	99% (1 error)
User-4	100%	97% (4 errors)
User-5	100%	100%

Table 24 Appendix H – Calibrations HTMV2 with Real Network Data

Calibration runs for qualifying Alternate Approaches with Synthetic Data

Users	Alternate Approaches % Accuracy
	1st Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	
	3rd Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	All K Order Markov (K=3)
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	PPM
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%

Table 25 Appendix H – Calibrations Alternate Markov Based Approaches with Synthetic Data

Calibration runs for qualifying Alternate Approaches with Real Network Data

Users	Alternate Approaches % Accuracy
	1st Order Markov
User-1	100%
User-2	99% (1 error)
User-3	100%
User-4	100%
User-5	
	3rd Order Markov
User-1	100%
User-2	100%
User-3	100%
User-4	100%
User-5	100%
	All K Order Markov (K=3)
User-1	100%
User-2	99% (1 error)
User-3	99% (1 error)
User-4	100%
User-5	100%
	PPM
User-1	100%
User-2	99% (1 error)
User-3	100%
User-4	100%
User-5	

Table 26 Appendix H – Calibrations Alternate Markov Based Approaches with Real Network Data

References

Agrawal, R., & Srikant, R. (1995). Mining Sequential Patterns. *Conference Data Engineering (ICDE'95)* (pp. 3-14) . Taipei, Taiwan: IEEE Computer Society.

Adamic, A. L. (1999). The Small World Web. *ECDL '99 Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries* (pp. 443–452). Paris, France: Springer-Verlag.

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

Adamic. A. L., & Huberman B. A. (2000b). Power-Law Distribution of the World Wide Web. *Science, 287,* 2115.

Balakrishnan, M., Mohomed, I., & Ramasubramanian, V. (2009). Where's that phone?: geolocating IP addresses on 3G networks. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (pp. 294-300). New York, NY: ACM.

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

Banse, C., Herrmann, D., & Federrath, H. (2012). 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* (Vol 376, pp. 235-248). Heraklion, Greece: Springer Berlin Heidelberg.

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

Barabasi, A. L., & Albert, R. (2002). Statistical Mechanics of Complex Networks. *Reviews of Modern Physics, 74*, 47-97.

Barrat. A., Barthelemy. M., & Vespignani. A. (2008). *Dynamical Processes on Complex Networks*. New York, NY: Cambridge University Press.

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

Begleiter, R., El-Yaniv, R., & Yona, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research, 22,* 385–421.

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

Bellovin, S., Leech, M., & Taylor, T. (2003). *ICMP Traceback Messages* (Report No. draft-ietf-itrace-04). Retrieved from the Internet Engineering Task Force (IETF) website: https://tools.ietf.org/html/draft-ietf-itrace-04

Bobier, B. (2007). *Handwritten Digit Recognition using Hierarchical Temporal Memory*. 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

Borges, J. (2000). *A data mining model to capture user web navigation patterns*. (Doctoral dissertation). Available from British Library. (OCLC: 556924443)

Borges, J., & Levene, M. (2004). *A dynamic clustering-basedMarkov model for web usage mining*. Unpublished manuscript. Retrieved from CoRR: Computing Research Repository. cs.IR/0406032.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., …Wiener, J. (2000). Graph Structure in the Web. *Proceedings of the 9th International World Wide Web conference on Computer Networks: The International Journal of Computer and Telecommunications Networking* (pp. 309-320). Amsterdam, Holland: North-Holland Publishing Co.

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

Carver, C. A. (2002). *Intrusion Response Systems: A Survey*. Unpublished manuscript. Department of Computer Science, Texas A&M University, College Station, TX.

Casado, M., & Freedman, M. J. (2007). 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)* (pp. 173-186). Cambridge, MA: USENIX Association Berkeley.

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

Chen, M. S., Park, J. S., & Yu, P. S. (1996). Data Mining for Path Traversal Patterns in a Web Environment. *Proceedings of the 16th International Conference on Distributed Computing Systems* (pp. 385-392). Hong Kong，China：IEEE Computer Society.

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

Clearly, J.G., & Witten, I.H., (1984). Data Compression using adaptive coding and partial string matching. *IEEE Transactions on Communications, 32*(4), 396-402.

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

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

Cormode, G., Korn, F., Muthukrishnan, S., & Wu, Y. (2008). On Signatures for Communication Graphs. *Proceedings of the IEEE 24th International Conference on Data Engineering* (pp. 189-198). Cancun, Mexico: IEEE Computer Society.

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

Domingos, P., & Hulten, G. (2000). Mining HighSpeed Data Streams. *ACM SIGKDD International Conference Knowledge Discovery in Databases* (pp. 71-80). Boston, MA: ACM.

Donley, C., Howard, L., Kuarsingh, V., Chandrasekaran, A., & Ganti, V. (2010). *Assessing the Impact of NAT444 on Network Applications* (Report No. draft-donley-nat444-impacts-01). Retrieved from the Internet Engineering Task Force (IETF) website: http://tools.ietf.org/html/draft-donley-nat444-impacts-01

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

Duch, W., Oentaryo, R. J., & Pasquier, M. (2008). Cognitive architectures: where do we go from here?. *Proceedings of the First conference on Artificial General Intelligence* (pp. 122-136). Memphis, TN: IOS Press Amsterdam.

Egevang, K., & Francis, P. (1994). *The IP Network Address Translator (NAT).* (Report No. RFC 1631). Retrieved from the Internet Engineering Task Force (IETF) website: http://www.ietf.org/rfc/rfc1631.txt

Feder, M., & Merhav, N. (1994). Relations between entropy and error probability. IEEE *Transactions on Information Theory*, *40*(1), 259-266.

Gates, C. (2009). Coordinated Scan Detection. *Proceedings of the 16th Annual Network and Distributed System Security Symposium* (NDSS'09) (pp. 153-165). San Diego, CA: Internet Society.

George, D., & Widrow, B. (2008*).* 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)

George, D., & Jaros, B. (2007). *The HTM Learning Algorithms*. Unpublished manuscript. Numenta Inc. Retrieved from http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&ved=0CDQQFjADOAo&url=http%3A%2F%2Fntebooks.googlecode.com%2Fsvn%2Ftrunk%2F%25D0%259A%25D0%25B8%25D0%25B1%25D0%25B5%25D1%2580%2FNumenta%2FNumenta_HTM_Learning_Algos.pdf&ei=Lxc1Ut6XC5LtrAGxv4HoBw&usg=AFQjCNGBqF8d-R0pXy7XUZ7vNzIIXsAAkg&sig2=QFRBe0JWeSYldDC9NZNk3g

Ghorbani, A. A., Lu, W., & Tavallaee, M. (2010). *Network Intrusion Detection and Prevention - Concepts and Techniques.* New York, NY: Springer.

Gray. J., Sundaresan. P., Englert. S., Baclawski. K., & Weinberger. P. J. (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.

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

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

Halabi, S. (2001). *Internet Routing Architectures (2nd Edition).* Indianapolis, IN: Cisco Press.

Han, J., & Kamber, M. (2006). *Data Mining Concepts and Techniques, Second Edition*. San Francisco, CA: Morgan Kaufmann.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C. (2000). Freespan: Frequent pattern-projected sequential pattern mining. *ACM SIGKDD International Conference Knowledge Discovery in Databases (KDD'00)* (pp. 355-359). Boston, MA: ACM.

Hang, Y., & Fong, S. (2010). Investigating the Impact of Bursty Traffic on Hoeffding Tree Algorithm in Stream Mining over Internet. *Proceeding of the 2nd International Conference on Evolving Internet (INTERNET)* (pp. 147-152), Valencia, Spain: Conference Publishing Services (CPS).

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

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

Herrmann, D., Banse, C., & Federrath, H. (2013). Behavior-based Tracking: Exploiting Characteristic Patterns in DNS Traffic. *Computers & Security*. Advance online publication. doi:http://dx.doi.org/10.1016/j.cose.2013.03.012

Hill, S., & Provost, F. (2003). The Myth of the Double-Blind Review? Author Identification Using Only Citations. *SIGKDD Explorations, 5*(2), 179-184.

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

Hills, S., & Nagle, A. (2009). Social Network Signatures: A Framework for Re-Identification in Networked Data and Experimental Results. *Computational Aspects of Social Networks, CASON '09* (pp. 88-97). Fontainbleu, France: IEEE Computer Society

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining TimeChanging Data Streams. *ACM SIGKDD International Conference Knowledge Discovery in Databases (KDD'01)* (pp. 97 - 106). San Francisco,CA: ACM.

Iváncsy, R., & Juhász, S. (2007). Analysis of Web User Identification Methods. *International Journal of Computer Science, 2*(3), 212-219.

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

Jiang, S., Guo, D., & Carpenter, B. (2011). *An Incremental Carrier-Grade NAT (CGN) for IPv6 Transition.* (Report No. RFC 6424). Retrieved from the Internet Engineering Task Force (IETF) website: http://tools.ietf.org/html/rfc6264

Jin, Y., Sharafuddin, E., & Zhang, Z.-L. (2007). Identifying Dynamic IP Address Blocks Serendipitously through Background Scanning Traffic. *Proceedings of ACM CoNext'07* (Article no. 4) . New York, NY: ACM.

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

Kumar, P., Krishna, P. R., & Raju, B. S. (2010). A New Similarity Metric for Sequential Data. *International Journal of Data Wharehousing and Mining, 6*(4), 16-32.

Kumpošt, M. (2007). Data Preparation for User Profiling from Traffic Log. *Proceedings of The International Conference on Emerging Security Information, Systems, and Technologies* (pp. 89-94).Valencia, Spain: Conference Publishing Services (CPS).

Kumpošt, M., & Matyáš, V. (2009). 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* (pp. 1-10). Berlin, Heidelberg: Springer-Verlag.

Lee, W., & Stolfo, S. (1998). Data mining approaches for intrusion detection. *Proceedings of the Seventh USENIX Security Symposium (SECURITY '98)* (Vol. 7, pp. 6-6). San Antonio, TX: USENIX Association.

Levene M., & Loizou G. (2003). Computing the Entropy of User Navigation in the Web. *International Journal of Information Technology and Decision Making, 2*(3), 459-476.

Liu, B. (2008). *Web Data Mining Exploring Hyperlinks, Contents and Usage Data*. Berlin, Germany: Springer-Verlag.

Liu, J.G., & Wu, W.P. (2004). Web Usage Mining for Electronic Business Applications. *Proceedings of the Third International Conference on Machine Leaming and Cyhemetics* (pp. 1314-1318). Shanghai, China: IEEE Computer Society.

Mankin, A., Massey, D., Wu, C.-L., Wu, S. F., & Zhang, L. (2001). On Design and Evaluation of "Intention-Driven" ICMP Traceback. *Proceedings of the IEEE Int'l Conf. Computer Comm. and Networks* (pp, 159-165). Scottsdale, AZ: IEEE Computer Society.

Manku, G. S., & Motwani, R. (2002). Approximate frequency counts over data streams. *Proceedings of the 28th Very Large Data Bases (VLDB) Conference* (pp. 346-357). Hong kong, China: VLDB Endowment.

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

Mobasher, B., Cooley, R., & Srivastava, J. (1999). Creating Adaptive Web Sites Through Usage-Based Clustering of URLs. *Proceedings of the 1999 IEEE Knowledge and Data Engineering Exchange Workshop* (pp. 19-25). Chicago, Illinois: IEEE Computer Society.

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

Mukund, D., & George, K. (2004). Selective Markov Models for Predicting Web-Page Accesses. *ACM Transactions on Internet Technology*, *4*, 163-184.

Nakhjiri, M., & Nakhjiri, M. (2005). *AAA and Network Security for Mobile Access: Radius, Diameter, EAP, PKI and IP Mobility*. West Sussex, England: John Wiley & Sons.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C. (2001). PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *International Conference Data Engineering (ICDE'01)* (pp. 215-224). Heidelberg, Germany: IEEE Computer Society.

Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys (CSUR*), *39*(1), Article 3.

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

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

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

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

Riesenhuber, M. & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience, 2*, 1019–1025.

Ron, D., Singer, Y., Tishby, N. (1996). The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning, 25*, 117-149.

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

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

Sarawagi, S. (2005). Sequence data mining. In Bandyopadhyay, S., Maulik, U., Holder, L. B., & Cook, D. J., *Advanced Methods for Knowledge Discovery from Complex Data* (pp. 153–187). London, England: Springer.

Sarukkai, R. R. (2000). Link prediction and path analysis using Markov chains. *Proceedings of the 9th international World Wide Web conference on Computer networks: the international journal of computer and telecommunications networking* (pp. 377–386). Amsterdam, Holland: Elsevier North-Holland.

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

Snoeren, A. C., Patridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T., & Strayer, W.T.. (2002). Hash-Based IP Traceback. *Journal of IEEE/ACM Transactions Networking, 10*(6), 721-734.

Song, D., Venable, P., & Perrig, A. (1997). *User recognition by keystroke latency pattern analysis*. Unpublished manuscript. Department of Computer Science, Berkley, CA.

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

Spiliopoulou, M., Mobasher, B., Berendt, B., & Nakagawa, M. (2003). A Framework for the Evaluation of Session Reconstruction Heuristics in Web-Usage Analysis. *INFORMS Journal on Computing, 15*(2), 171–190.

Srikant, R., & Agrawal, R. (1996). Mining Sequential Patterns: Generalizations and Performance Improvements. *Proceedings 5th International Conference Extending Database Technology (EDBT'96)* (pp. 3-17). Avignon, France: Springer-Verlag.

Srivastava, J., Cooley, R., Deshpande, M., & Tan, P.-N. (2000). Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *ACM SIGKDD Explorations Newsletter, 1*(2), 12-23.

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

Watts, D. J., & Strogatz, S. H. (1998). Collective Dynamics of Small World Networks. *Nature Journal of Science, 393*, 440-442.

Xie, Y., Yu, F., & Abadi, M. (2009). De-anonymizing the Internet Using Unreliable IDs. *ACM SIGCOMM Computer Communication Review* (pp. 75-86). Barcellona, Spain: ACM.

Xing, Z., Pei, J., & Keogh, E. (2010). A brief Survey on Sequence Classification. *ACM SIGKDD Explorations, 12*(1), 40-48.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining Closed Sequential Patterns in Large Datasets. *SIAM International Conference Data Mining (SDM'03)* (pp. 166-177). San Francisco, CA: SIAM.

Yang, Yinghui. (2010). Web user behavioral profiling for user identification. *Decision Support Systems, 49*(3), 261-271.

Zaki, M. J. (2001). SPADE*:* An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning, 32*(1-2), 31-60.