Mimicking Human Behavior in Shared-Resource Computer Networks

Brian Ricks
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bwr031000@utdallas.edu

Bhavani Thuraisingham
Computer Science Department
University of Texas at Dallas
Richardson, USA
Email: bxt043000@utdallas.edu

Patrick Tague
ECE Department
Carnegie Mellon University
Moffett Field, USA
Email: tague@cmu.edu

***Abstract*—Among the many challenges in computer network trace data collection is the automation, or mimicking, of human users in situations where humans-in-the-loop are either impracticable or not possible. While client-side human behavior has been automated in various static settings, autonomous clients which dynamically change their behavior as the environment changes may result in a more accurate representation of human behavior in captured network trace data, and thus may be better suited for problems in which humans-in-the-loop are important. In this work, we set out to create dynamic autonomous client-side behavioral models, which we call agents, that can interact with the network environment in much the same way that humans do, and are scalable in shared-resource environments, such as emulated computer networks. We show through multiple experiments and a web crawling case study on an emulated network that our agents can mimic interactive human behavior, and do so at scale.**

I. Introduction

Network trace data which adequately captures behavior required for research problems is often scarce or non existent [1], forcing researchers to either produce their own data, or perhaps change research directions. While the latter is usually undesirable, the former poses many challenges on its own, and may become a problem in its own right. A researcher tasked with collecting or generating relevant network trace data may need to solve problems such as locating a viable computer network in which data can be collected from, and assessing if such a network is adequate to capture behavior related to the research problem [2]. Even if a suitable target computer network exists and data can be collected from, further complications arise if human user behavior needs to be included in the capture, such as policy to permit such capture, how human users react to environments in which they know their data is being captured [3], and issues that will naturally arise if *specific* human behavior needs to be represented in the captured data.

In this work we aim to take client-side human users out of the loop and generate realistic network trace data by using dynamic *agents* to replace human users. An agent is an autonomous client which can interact with its environment, dynamically changing its behavior based on what it sees, or senses [4]. The idea is to create agents which can adapt their behavior when interacting within a network, in much the same way that humans do. Our goal is to *mimic* specific human behavior given the same environmental conditions that would be present to human users.

As a rolling example presented throughout this paper, we consider the use case of video streaming services, such as YouTube. Sequences of videos which a typical human user may watch on a video streaming service is heavily influenced by what the recommendation algorithm features to that user [5], [6]. Recommendations from the algorithm may be based on many different factors, such as *viral* videos - videos that are currently popular on the service, and new videos which other creators recently uploaded. To properly mimic human users, an autonomous client needs to not only sense these factors, but use them in the decision making process.

This work focuses on a specific type of computer network in which humans-in-the-loop are impracticable or impossible: shared-resource emulated computer networks. An emulated computer network comprises one or few physical computers hosting a large-scale computer network by means of virtualizing network hosts and simulating the physical links between them. Each virtual network host must share the resources of the physical computer among all other virtual hosts. Emulated computer networks are important in areas such as academic labs in which physical network testbeds are often difficult to build due to time and financial constraints [7].

Our main contributions in this work are as follows:

- The development of agents which can make decisions, in part, based on what is going on around them, similar in nature to how humans often make decisions.
- An agent architecture that is conducive for deployment in shared-resource computer networks, which is important for researchers who need to generate or collect network trace data incorporating human user behavior, but without the proper physical resources to perform the task.

The rest of this paper is structured as follows. In Section II we discuss relevant background and related work. Section III discusses our approach to mimicking human behavior as agents, including interactive human behavioral models and agent architecture. Section IV presents our experiments and results, with relevant discussion. More specifically, Section IV-B presents experimental results involving our agent-based autonomous clients, and Section IV-C presents our shared-resource scalability results. We then discuss limitations

of our approach and framework, along with future work, in Section V, and we conclude the paper in Section VI.

II. Preliminaries

A. Network Testbed Methodologies

Computer network testbeds can roughly be divided into three categories: physical networks, emulated networks, and simulated networks. Physical networks, as the name suggests, are networks which comprise physical hosts, links, and other infrastructure [8], [9]. While physical networks are ideal for realistic network trace data generation, they suffer from limitations which particularly affect academic labs. Cost and time to build these networks often rule them out for data collection purposes, and space considerations also can be a limiting factor. Physical networks also do not scale well in terms of reliability.

On the other end are simulated networks, which enable simulation of an entire physical network on a single physical host, and often at scale even on modest hardware [10], [11]. However, because network links and the entire network stack is simulated, accompanying assumptions and simplifications may hinder the generation of data realistic enough for a given problem. Also to note, experimentation in real-time is typically not possible as network simulators often use event-queues to dispatch network events as opposed to timers [10].

A middle-of-the-road approach uses network *emulation* [12], [13], [14], which offers advantages present from both physical and simulated networks. Like physical networks, emulated networks run the same network stacks and host services/protocols. Like simulated networks, an emulated network can run entirely within a single physical host, which is where the limitations come into play. Hosts within an emulated network each run in a virtualized environment [15], and links between them must be simulated as the network itself is not physical. However, as long as the physical host has enough computational and memory resources to properly host an emulated network, resource contention should not be an issue. Also, provided that the emulated network is simulating wired links, such as Ethernet links, the simplifications present with such links should not impede the generation and collection of realistic network trace data.

The approach presented in this paper will utilize network emulation to achieve the goals laid out in Section I.

B. Network Traffic Automation

Physical and emulated networks require either real humans to interact with the network, or a form of automation to emulate human user behavior. Bringing real human beings into the loop is challenging not only in terms of scale, but also in terms of behavior preservation [3]. Therefore, while not ideal, it may be more practical to automate client-side user behavioral patterns.

Prior work utilizes varying approaches to automate client-side user behavior, such as using previously captured network trace data to build autonomous clients [16], or utilizing expert-defined profiles to drive autonomous behavior [17]. Our prior work automated client-side user behavior in a *static* nature, with behavior that is independent of the network environment [7]. In this work, we would like to enable *dynamic* client-side behavior that is *interactive* with the network. This requires an approach to enable our client-side agents to interact with the environment (network) in a manner that is scalable in a shared-resource computer network, such as an emulated network.

Building off our prior work, we augment the eMews network traffic automation and experiment management framework [7]. eMews enables user automation through the use of behavioral models, called *services*, while providing a verbose logging environment to monitor a running network experiment. Also of importance, the framework is designed to run on shared-resource networks, enabling many services to run at scale. We extend eMews to support our agent-based system, which already provides much of the lower level network and centralized communication required for our implementation.[1]

C. Intelligent Agents

An intelligent agent typically uses on-board sensors to collect or sense information, or *evidence*, from the environment within which it interacts, and uses this evidence to guide its decision process in terms of the next action to take [4]. For any given action performed by an agent, evidence within the environment may change. For example, consider an autonomous vacuum cleaner which cleans any area it finds is dirty. The vacuum cleaner is the agent, cleaning is an action the agent takes, and sensing whether an area is clean or dirty is evidence collection from the environment. Evidence can change when a dirty area is cleaned.

In our domain, the network itself serves as the environment, and our client-side agents manifest as programs within network hosts. Agents make decisions, in part, on network state based evidence, which could comprise anything ranging from host input traffic to the actions of other agents from other network hosts. In terms of our viral video example, an agent could base its decision of which video to click, in part, on any viral videos the agent is made aware of. This strategy departs from approaches used in similar domains, such as link prediction [18], [19], in which decisions in the present are often determined from prior decisions.

eMews static client-side automation services (Section II-B) decide which actions to perform by distribution sampling, whose parameters are mostly determined from expert knowledge [7]. One exception is the *SiteCrawler* service, a web site link crawler, which for any currently loaded web page containing a set of links, decides which link to click by sampling from a truncated normal distribution whose parameters are determined by the number of links which comprise the currently loaded page. This induces an implicit dependence on the currently loaded web page when the SiteCrawler service is making a decision on which link to follow, which arguably may be considered dynamic behavior. However, in this case,

[1] eMews can be downloaded here: http://mews.sv.cmu.edu/research/emews/

the service itself is bounded to making a decision within the space of links on the currently loaded page, and that decision is strictly by sampling. An intelligent agent, on the other hand, may factor in additional information it senses from the environment when making a decision.

III. Mimicking Human Behavior

We start this section by introducing the underlying architecture necessary for our agent-based approach to work. Then we discuss our agent implementation, from the perspective of a web crawling agent, which is a generalization of the video streaming example presented in Section I. The overall goal of mimicking human behavior using this approach is to provide such behavior that can be captured in network trace data.

A. Approach Architecture

Our approach utilizes a common concept with agents known as *ask/tell*, in which an agent can sense (ask) the environment to obtain *evidence* for decision making, and also can update (tell) the environment directly about its actions, decision processes, etc., which we call *observations*. Giving observations directly to the environment is feasible in our domain by network message passing. The environment itself lives as an EMEWS server module on a single virtual host, supporting multiple simultaneous ask/tell requests from agents in a centralized manner.

Whenever an agent gives observations to the environment (tell), the observations are processed by the environment for evidence generation. Newly generated evidence is stored within the environment for agents to query (ask).

Exactly how new evidence is produced is dictated by logic which form part of the schema for an experiment. For example, if agents are crawling a video streaming server, these agents may give to the environment observations regarding which video they are currently crawling. The environment may contain logic which dictates to produce evidence of a viral video if the number of crawls to that video reaches a specific threshold within some time interval. This evidence could then be queried by an agent to guide its decision as to which video to crawl next. This specific example implies that evidence is not necessarily produced each time an agent gives observations to the environment, but rather produced when specific sets of observations are given by perhaps multiple agents and perhaps over a varying interval of time.

One subtlety with how our approach handles evidence deals with expiration. For example, it wouldn't be very realistic for evidence of a viral video to remain indefinitely. This implies that whenever an agent asks the environment for evidence, the evidence returned is relevant at the time of asking.

B. Web Crawling Case Study

We demonstrate our approach in terms of an agent for web crawling. A web crawling agent connects to a web server (web site) and proceeds to visit pages. Each page embeds *links*, which when clicked load another page corresponding to the link clicked. The agent will then pick a link on the newly loaded page to click, and the pattern continues. A sequence of clicked links comprises a page crawl. The web crawling activity is sub-divided into *sessions*, in which a single session comprises a series of page crawling from a single web server. During a session, the task of an agent is to decide a link to click given a current page that the agent is visiting (crawling).

For any given page that an agent is visiting, it will be presented with n links contained on the page to choose from to click next. The agent can also query evidence from the environment by *type*. For example, a list of links which are currently viral constitutes one type of evidence, and a query for this type will return all the links on the current page which are viral. Making the closed world assumption that a link present on the page but not present in the list is non-viral, and is in itself evidence, the total pieces of evidence pertaining to viral and non-viral links is equal to the number of links on the page, or in other words, $|page\_links| = |viral\_link\_evidence| = n$. The evidence types we use for our model adhere to this property.

Our web crawling agent model supports two types of evidence: viral links, and previously visited links. Viral links are analogous to the concept of viral videos presented in our rolling example from Section I, and we generalize this concept for links to include any link which is recommended to an agent, or in other words, provided to an agent as evidence. Previously visited links are links which an agent has already visited during the current web crawling session. Note that both types of evidence contain n pieces, one for each page link, for a total of $2n$ pieces of evidence in total.

1) Agent Environment Overview: As mentioned briefly in Section III-A, the agent environment is tasked with producing evidence that agents can then ask for. For the evidence type of viral videos, the environment determines whether a link is viral by counting clicks to that link among all the agents. The environment will know when an agent clicks on a link because the agent will tell the environment about its actions. Whenever the total count of clicks for any link on any specific web server reaches a threshold, given in the schema, then the environment will produce new evidence of the viral link. This is accomplished by using a unique identifier to index the link and adding it to a list of viral links present on the currently visited page. The environment knows which indices map to which links as agents tell the environment about which web server they are crawling and links clicked in terms of these same indices.

The visited links evidence type does not require the environment, as agents keep track of their visited links for a given session.

2) Agent Model: The key goals of our agents are to provide for easy configuration of desired human behavior, and in a manner conducive for deployment in a shared-resource environment, possibly running hundreds or thousands of agents concurrently. Based on the methodology proposed in Ricks et al. [7], our agent model should comprise of logic blocks that are in themselves not time-consuming to execute. In that spirit, we present the following model.

At the heart of the agent is a model which determines which link to click next on the currently visited web page, given evidence. The model is represented as a discrete joint probability distribution D, in which a random variable X captures the state space of possible agent actions $x \in X$, and each piece of evidence e_i, $1 \leq i \leq mn$ is represented as a random variable E_i in D. Here, m is the number of evidence types, and n is the number of variables which comprise a specific type. So for mn pieces of evidence, we have a joint distribution comprising $mn+1$ random variables. The goal of the agent is to sample from D the next action $x \in X$, given evidence $\mathbf{e}$.

If we represent the joint distribution D as a table, then it will contain $|X| * |E|^{mn}$ parameters, where $|X|$ is the number of states (cardinality) for variable X, and $|E|$ is the cardinality for each evidence variable E_i, assuming each variable E_i has the same cardinality. This can become intractable quite fast. For example, if the agent has 10 links to choose from ($|X| = 10$), and each piece of evidence represents either a specific link that has gone viral (either the link is viral or not: $|E_i| = 2$) or a visited link (again, the link is either visited or not), then we have a total of $10 * 2^{2*10} = 10,485,760$ parameters. $10,485,760$ parameters would clearly be too many to specify, and assuming each parameter is stored as a 32 bit integer, a single agent would need 40 MB in the worst case just to store the model, not to mention inference overhead. As the number of links increase, the memory requirements increase exponentially.

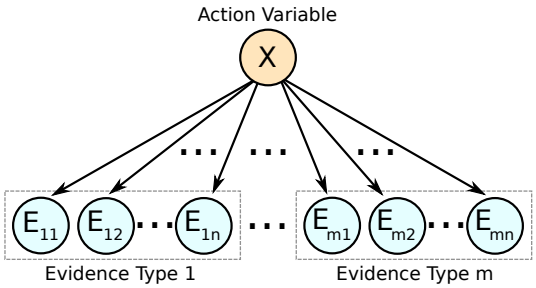

Fig. 1. The factorization strategy used in our agent model. Evidence is categorized as types, with each type comprising $|X|$ random variables. Variable X's state space are the possible actions an agent can take, which for our use case are the clickable links on the currently visited page.

Given these challenges and our shared-resource constraint, we opt to use a common assumption of independence among our evidence variables, known as the *naive Bayes assumption*, which states that every evidence variable is independent of every other evidence variable given an action[2]. This allows us to factorize our joint probability distribution as a Bayesian network [20] representing a naive Bayes structure, reducing parameter space to only those required to specify each joint conditional probability distribution of each evidence variable given an action (Figure 1). Taking the example above, this assumption reduces the number of parameters we need to define from $10,485,760$ to $|X| + (mn)|E| * |X| = 10 + 20 * 2*10 = 410$ parameters, a drastic reduction, but still too many to specify manually.

[2]This assumption is not always accurate to make, but tends to work well in practice.

To work around the need to specify all parameters, we instead define hyper-parameters, and use those to generate the model parameters. We define three strictly positive hyper-parameters that need to be specified by the researcher collecting the data: α, the relative strength that an agent should click a link that is viral, β, the relative strength that an agent should click a link it has already visited in a given crawling session, and δ, the relative strength that a set of links should be preferred to click over all others. The term 'strength' refers to a number that defines a ratio between specific probabilities to all others in a distribution. For example, in a given distribution, a strength of 10 means that probabilities in which the strength is applied will be scaled at a factor of 10 relative to all other probabilities in the distribution.

The hyper-parameters α and β are used to calculate the conditional probabilities for the evidence variables, and correspond to the evidence types of viral links and visited links, respectively. The hyper-parameter δ is used to calculate the probabilities for our prior distribution X.

Given that a link j was clicked, the conditional probability that link j was given as positive evidence y (for example, link j is viral or previously visited), is defined for all evidence variables E_i as:

$$P(E_i = y|X = x_j) = \frac{\rho}{\rho + |X| - 1}, \tag{1}$$

in which ρ is the hyper-parameter (for example α or β). Once calculated, only two parameters are stored per evidence type, $P(E_i = y|X = x_j)$, defined above, and $P(E_i = y|X \neq x_j)$, for all evidence variables E_i, which is calculated by:

$$P(E_i = y|X \neq x_j) = \frac{1}{\rho + |X| - 1}. \tag{2}$$

To incorporate additional types of evidence in this way, we would need one additional hyper-parameter for each type of evidence, and $|X|$ additional random variables.

Note that for any evidence variable E_i and clicked link j, $P(E_i = y|X = x_j) + (|X| - 1) * P(E_i = y|X \neq x_j) = 1$. This a side effect of normalizing the probabilities (Equations 1 and 2) to have the correct ratio based on the given strength hyper-parameters. This is dependent on the number of links, as an agent ultimately samples a link to click, and the strength ratios need to be correct in the posterior (Section III-B3).

Our prior belief that a specific link will be clicked is represented by X, the action variable. The probabilities for this distribution are calculated using hyper-parameter δ and a set of links $\{L\}$ which are preferred to be clicked by the agent, for each preferred link j, using the following:

$$P(X = x_j) = \frac{\delta}{\delta * |\{L\}|' + |X| - |\{L\}|'}, \tag{3}$$

where $|\{L\}|'$ is the cardinality of preferred links which are present in the current page. For each non-preferred link k, we calculate the probability using the following:

$$P(X = x_k) = \frac{1}{\delta * |\{L\}|' + |X| - |\{L\}|'}, \quad (4)$$

where $\sum_j 1 + \sum_k 1 = |X|$. This results in two parameters that are stored to define X.

The proof that X is a valid probability distribution follows that $|\{L\}|' * P(X = x_j) + (|X| - |\{L\}|') * P(X = x_k) = 1$, where j is a preferred link and k is not, and $0 \leq P(X = x_i) \leq 1$, for all links i.

Each time the agent crawls to a new page, it is presented with a new set of links to choose from. This implies that the joint distribution to sample the next link could have a different value for $|X|$, also implying that the number of variables may change. Thus, we recompute the distribution parameters with the new value of $|X|$ every time a new page is crawled. Note that the hyper-parameters act as constants, and do not change value during a crawling session.

To complete the agent model specification, two additional discrete univariate distributions are used to sample the amount of time that an agent waits between web crawling sessions, and the amount of time an agent waits between clicking links during a session. Each distribution requires two parameters, a lower and upper bound, and values within this bound are sampled uniformly.

3) Agent Model Inference: An agent decides the next link to click by sampling from the discrete joint probability distribution D described in Section III-B2. The query we need to perform proper sampling is $P(X|\mathbf{e})$, the distribution of X given our evidence $\mathbf{e}$, called our posterior. For a naive Bayes model, the standard way of accomplishing this is by using the chain rule for probability:

$$P(X = x|\mathbf{e}) = \gamma P(X = x) \sum_{i=0}^{|\mathbf{E}|} P(E_i \backsim \mathbf{e}|X = x),$$

for all $x \in X$, where γ is a normalization constant, $|\mathbf{E}|$ is the number of evidence variables, and $\backsim$ meaning that the variable E_i is consistent with evidence $\mathbf{e}$.

Normalization constant γ is equivalent to $P(\mathbf{e})$, the probability of evidence, and gives us valid probabilities for our posterior. Once our posterior probabilities are normalized, we then sample a value $x \in X$ directly from the posterior by sampling a continuous value r uniformly within the interval $[0, 1)$, and building up a cumulative distribution function (CDF) of the posterior in which each step in the CDF represents a distinct $x \in X$. Whenever the cumulative value of the CDF becomes greater than r, we take as the sample the value x at that step.

IV. Experiments

In this section we discuss our experimental methodology and present results in terms of our agent behavioral patterns and scalability.

A. Methodology

Our agent experiments (Section IV-B) focus on specific use cases of real human behavior that a researcher may want to capture in network trace data. This distinction is important as we are not trying to generalize human behavior, but rather mimic specific behavior so it can be captured. We also present scalability results, which are important in showing that our agents can scale on shared-resource hardware.

To this end, we used a Dell Inspiron laptop to run our experiments, with the following configuration: Intel 7th generation i7 dual-core processor with hyperthreading (4 threads), 16GB RAM, and running Ubuntu 18.04 LTS. A lower-end laptop such as this one was chosen purposefully to demonstrate that even modest resources can be used to successfully generate relevant network trace data. CORE 5.2 served as our emulated network framework, running eMews 0.4, which includes our extensions to enable agents and the environment.

The environment schema for the viral evidence type is as follows: the number of times a link needs to be clicked before going viral is set to 10, and the interval of time in which a viral link would remain viral is set to 60 seconds. The web server hosts a single page with 18 links that loop back to the same page. This way we eliminate any dependencies induced from partitioning the links over multiple web pages.

B. Results - Agent Experiments

We start with agents that utilize only some of the evidence available to them, and build up to agents which not only utilize all available evidence, but also have their own behavioral preferences on link selection. We present the results in terms of specific use cases from the video streaming service domain introduced in Section I.

1) Use Case 1: Open-Minded and Curious: Figure 2 illustrates an agent in action crawling page links. This agent is configured to only utilize evidence of the viral link type. Thus, when no links are viral, it displays a uniform pattern of link selection, basically picking links at random. However, during the period of time illustrated in the figure by a red box, the link at index 1 was viral. During this time period, the agent clearly had a preference for this link, as shown in Figure 2. This is to be expected, as the agent has no aversion to clicking on links it has already visited in the same web crawling session.

Human behavior that this agent would mimic includes bored humans randomly clicking videos but preferring the videos recommended to them, or humans that predominantly click on recommended videos, the latter being the mimicked behavior if the link space was larger and more agents were simultaneously clicking links. This behavior is also characterized by a tendency to re-watch the same video multiple times if it is not viral, which humans may do if the video has a high re-watch value, such as music videos.

2) Use Case 2: Closed-Minded and not Curious: Figure 3 illustrates an agent with link preferences, but no other utilization of evidence. Perhaps unsurprisingly, this results in a large skew of clicked links to the preferred links, which for this agent are the links at indices 5 and 10. While some of

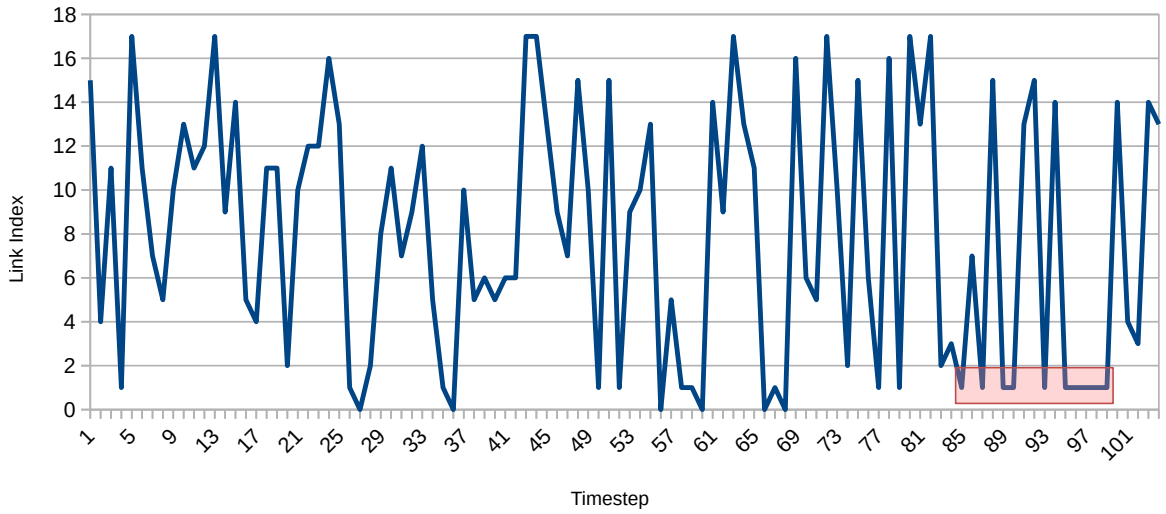

Fig. 2. Sequence of link clicks by an agent configured to utilize viral link evidence. Its parameters are configured as: $\alpha = 10.0$, $\beta = 1.0$, $\delta = 1.0$. The values for β and δ render the agent blind to previously clicked links, and gives the agent no prior link preference. The red box in this figure illustrates the time period in which the link at index 1 went viral. The agent clicked on that link 9 times during the short time period in which it was viral, and 11 times when the link was not viral, which encompasses a much larger interval of time.

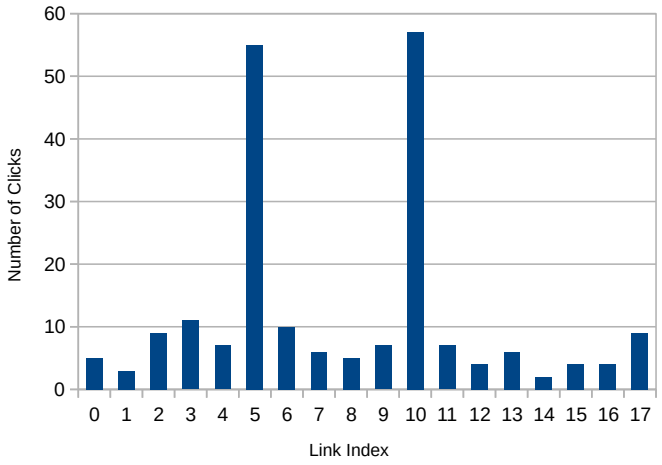

Fig. 3. Distribution of link clicks by an agent with the following parameters: $\alpha = 1.0$, $\beta = 1.0$, $\delta = 10.0$ with link indices 5 and 10 as preferred. Note that the α and β parameter values render the agent blind to all evidence. The preferred links are noticeable in the figure as having the most clicks between them, dominating the other links.

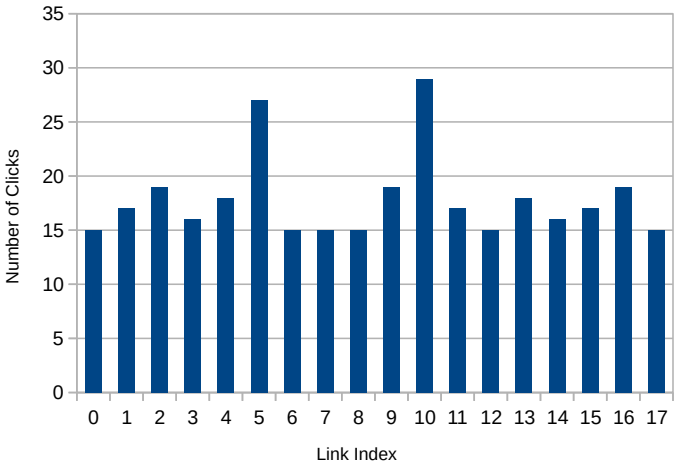

Fig. 4. Distribution of link clicks by an agent with the following parameters: $\alpha = 10.0$, $\beta = 0.01$, $\delta = 10.0$ with link indices 5 and 10 as preferred. The preferred links are noticeable in the figure as having the most clicks between them, but is not as dominating as the preferred link clicks from the agent represented in Figure 3.

the links went viral during this agent's crawling sessions, that evidence would have had no impact on the decisions made by the agent.

Human behavior that this agent would mimic comprises picky humans that know which videos they want to watch, and do not pay attention to the recommended videos. This behavior is also characterized by a natural aversion to watching the same video multiple times, which is common place for types of videos that do not have a high re-watch value, such as news clips.

3) Use Case 3: Closed-Minded and Curious: Referring to Figure 4, we now move to an agent that utilizes all available evidence. Notice that link indices 5 and 10 have the most clicks, which makes sense as these are the agent's preferred links to click. Link index 2 did go viral during this run, but the number of clicks for this index was not as high as indices 5 or 10. This intuitively makes sense, as this agent is configured with parameter $\beta = 0.01$, giving the agent a relatively high aversion to clicking on links it has already visited, in spite of any of them going viral. The higher click count for link index 2 is most likely due to the agent leaving the web server and coming back to start to a new session; it finds link index 2 is still viral, and it had not clicked it yet during the new web crawling session.

Human behavior that this agent would mimic includes humans that know which videos they would like to watch, but are curious about the recommended videos as well, and

Agent Count	Memory Usage (Avg)	CPU Usage (Avg)
12	600MB	3%
30	1.52GB	7%
60	2.83GB	15%
126	6.14GB	34%
216	9.91GB	49%
235	10.74GB	51%
331	13.36GB	57%

TABLE I
SCALABILITY RESULTS OF AGENT DEPLOYMENT IN SHARED-RESOURCE NETWORKS. EACH AGENT RAN ON ITS OWN VIRTUAL HOST WITHIN THE EMULATED NETWORK. AVERAGE MEMORY AND CPU USAGE WERE MEASURED FROM THE PHYSICAL HOST, SO RESOURCE USAGE FROM OTHER PROCESSES ARE ALSO PRESENT.

will consider them when they are presented. A natural aversion is present to re-watching videos in general, which increases as the link space increases.

4) Agent Interaction: When multiple agents are running concurrently on a network, their individual actions may influence the actions of other agents. For example, when a link goes viral, that evidence alone may influence the decisions of some agents, but the clicks to get the link to become viral in the first place may have been the act of other agents. This type of indirect interaction among agents is one of the key motivations for this agent-based approach, and we capture such a scenario in Figure 5. Here we have the distribution of link clicks among five agents running concurrently. No two agents exhibit the same web crawling behavior, but some interaction based dependencies are present. The agent represented by the blue bars (blue agent) in Figure 5 has a very strong preference for clicking on links at indices 2 and 6. This behavior helped drive the link at index 2 to become viral, which then influenced other agents, specifically the yellow and green agents, to click on this link, as illustrated by a click count of 6 and 3 for these agents, respectively.

C. Results - Scalability

The goal of these experiments were to measure how well our agents scale in a shared-resource computer network environment. Referring to Table I, we were able to reach 331 concurrently running agents before memory became a concern. Compared to the results given for static services [7], we doubled the RAM available to the emulated network, but disabled swap space, so the total amount of memory remained equal. Our agents actually used slightly less memory than the static services. We believe this is due, in part, to optimizations we employed to reduce the overall memory footprint of our agents and environment structure, in anticipation of the inevitable overhead induced from our architecture on the original EMEWS framework.

While we were hoping for even better scalability results, running 300+ agents concurrently on a laptop we feel is still adequate for many experimental scenarios in which a cheap laptop would suffice as the physical host for the emulated network. On physical hosts with more resources available, such as a mid-range server, running thousands of agents concurrently should be possible.

D. Discussion

One question that may loom at the forefront is whether the agent-based behavior presented here can adequately mimic a wide enough range of real human behavior to be useful for capture. While this is to a degree subjective, it is worthwhile to point out that depending on configuration, agents in the web crawling use case behave in a manner consistent with how recommendation algorithms expect human users to behave. And these algorithms have been improved on for well over a decade. If they didn't work well with enough human users, they would most likely not be used at all.

The naive Bayes assumption may not be reasonable for all types of evidence an agent asks for. The reduction in the parameter space, however, is necessary in our shared-resource network environment, and the evidence types we present in this paper are reasonable for the naive Bayes assumption. For example, if multiple links are viral at one given time, a user can still only click one of them, and without any additional context as to which viral link a user may click, we can simply assume that the user may click any viral link with the same probability, which is what our model captures.

It is to note that exactly how the agent model hyperparameters are set is based on the network trace data that is desired to be captured. One advantage of using autonomous agents as opposed to real humans is specific human-like behavior can be produced by our agents that may be difficult to get real humans to produce without telling said humans how to act, which defeats the purpose of using real humans. This is also one reason why we do not compare real human behavioral trials to our agents, as the use cases themselves are considered ground truth.

V. LIMITATIONS AND FUTURE WORK

Mimicking real human behavior is a difficult problem due to the sometimes unpredictable way that humans act. There may be cases in which our agents do not mimics humans in a way desirable for some specific data capture. This is part of the nature of the problem in general, as there is no approach which will successfully mimic any human being. Rather, our goal is to mimic some common human behavioral patterns in a dynamic manner, for specific use cases.

This work is limited to only one agent type, the web crawling agent. However, the same basic approach coupled with new environment schemas can be applied to produce other agent types.

Future work includes producing additional agent types which can mimic human behavior for other tasks outside of web crawling, and further increasing the scalability of our approach architecture. We also plan to use our approach to generate network trace data for specific networking problems in which currently available data is sparse.

VI. CONCLUSION

In this work we set out to automate dynamic human client-side behavior in a network environment. Our agent-based approach, as shown in Section IV-B, mimics human-like

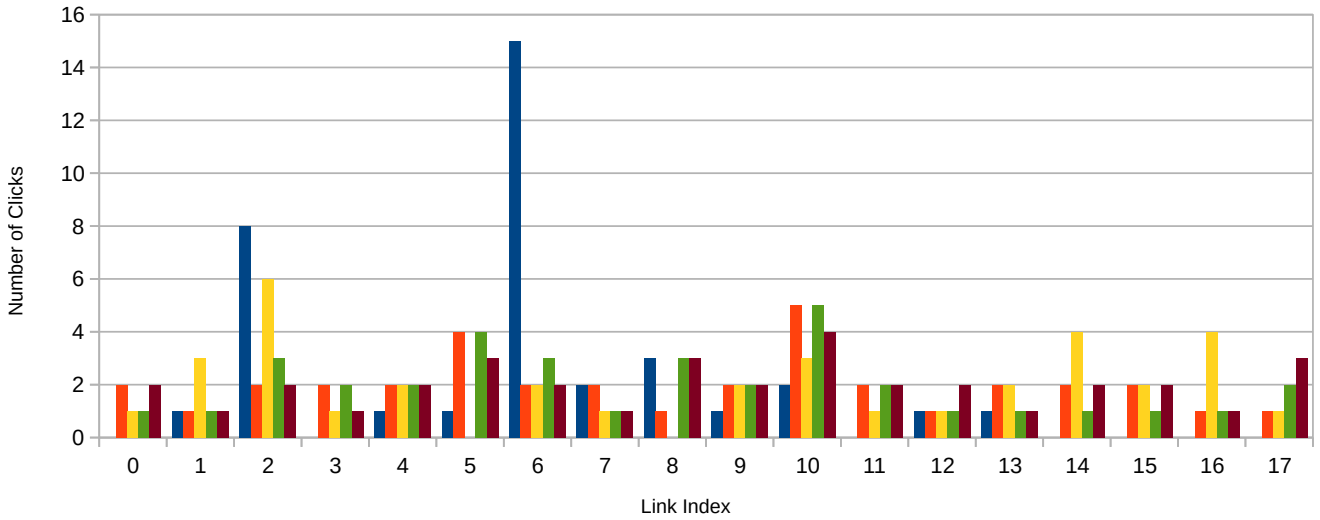

Fig. 5. Distribution of link clicks by five agents. Each agent had its own link preferences and willingness to click on viral links. Each of these agents produced unique behavior, comparable to how human behavior often differs from person to person. For example, the agent represented by the blue bars (blue agent) has a very strong preference for clicking on links at indices 2 and 6, as illustrated by a click count of 8 and 15, respectively. This behavior helped drive the link at index 2 to become viral, which then influenced other agents, specifically the yellow and green agents, to click on this link.

behavior that is dynamic both among the network environment and among other agents, and in a manner that is conducive for deployment within shared-resource networks and at scale, even on modest physical hosts. We hope that this work provides a way for researchers who need realistic human behavior in their network trace data to capture such data from the comfort of their own labs.

References

[1] B. Ricks, B. Thuraisingham, and P. Tague, "Lifting the Smokescreen: Detecting Underlying Anomalies During a DDoS Attack," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 130–135.

[2] H. Pucha, Y. C. Hu, and Z. M. Mao, "On the impact of research network based testbeds on wide-area experiments," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 133–146. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177097

[3] M. Bateson, D. Nettle, and G. Roberts, "Cues of being Watched Enhance Cooperation in a Real-World Setting," *Biology Letters*, vol. 2, no. 3, pp. 412–414, 2006.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[5] S. K. T. Lam, D. Frankowski, and J. Riedl, "Do you trust your recommendations? an exploration of security and privacy issues in recommender systems," in *Emerging Trends in Information and Communication Security*, G. Müller, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 14–29.

[6] O. Celma and P. Lamere, "If you like the beatles you might like...: A tutorial on music recommendation," in *Proceedings of the 16th ACM International Conference on Multimedia*, ser. MM '08. New York, NY, USA: ACM, 2008, pp. 1157–1158.

[7] B. Ricks, P. Tague, and B. Thuraisingham, "Large-Scale Realistic Network Data Generation on a Budget," in *19th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018.

[8] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building planetlab," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 351–366. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298489

[9] E. Jaffe, D. Bickson, and S. Kirkpatrick, "Everlab: A production platform for research in network experimentation and computation," in *LISA*, 2007.

[10] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Brussels, Belgium: ICST, 2008, pp. 60:1–60:10.

[11] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2

[12] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 271–284, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844154

[13] J. Ahrenholz, "Comparison of core network emulation platforms," in *2010 - MILCOM 2010 Military Communications Conference*, Oct 2010, pp. 166–171.

[14] J. Ahrenholz, T. Goff, and B. Adamson, "Integration of the core and emane network emulators," *2011 - MILCOM 2011 Military Communications Conference*, pp. 1870–1875, 2011.

[15] G. Calarco and M. Casoni, "On the effectiveness of linux containers for network virtualization," *Simulation Modelling Practice and Theory*, vol. 31, pp. 169 – 185, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X1200161X

[16] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 111–122. [Online]. Available: http://doi.acm.org/10.1145/1159913.1159928

[17] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357 – 374, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404811001672

[18] J. Zhu, J. Hong, and J. G. Hughes, "Using markov chains for link prediction in adaptive web sites," in *Soft-Ware 2002: Computing in an Imperfect World*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 60–73.

[19] A. L. Montgomery, S. Li, K. Srinivasan, and J. C. Liechty, "Modeling online browsing and path analysis using clickstream data," *Marketing Science*, vol. 23, no. 4, pp. 579–595, 2004.

[20] P. A. Darwiche, *Modeling and Reasoning with Bayesian Networks*, 1st ed. New York, NY, USA: Cambridge University Press, 2009.