Modeling How Thinking About the Past and Future Impacts Network Traffic with the GOSMR Architecture

Kevin Gold
MIT Lincoln Laboratory
Lexington, MA
kevin.gold@ll.mit.edu

Zachary J. Weber
MIT Lincoln Laboratory
Lexington, MA
zweber@ll.mit.edu

Ben Priest
MIT Lincoln Laboratory
Lexington, MA
benjamin.priest@ll.mit.edu

Josh Ziegler
Air Force Institute of Technology
Wright-Patterson AFB, OH
Joshua.Ziegler@afit.edu

Karen Sittig
Massachusetts Institute of Technology
Cambridge, MA
kasittig@mit.edu

William W. Streilein
MIT Lincoln Laboratory
Lexington, MA
wws@ll.mit.edu

ABSTRACT

We present the GOSMR architecture, a modular agent architecture designed to actuate web browsers and other network applications, and demonstrate the importance of modeling how users think about the past and future in accurately modeling network traffic. The architecture separates the hierarchical generation of goals and incentives (Behaviors) from hierarchical implementations of their pursuit (Actions). Cognitive aspects modeled include the hyperbolic discounting of future payoffs, the chance a user forgets a task, and the ability of the user to defer tasks for later. The system also uses a logical grammar to allow agents to communicate Beliefs and delegate Actions. Using records of weekend Virtual Private Network traffic from over three thousand users at a medium-scale enterprise, we provide evidence for the importance of the forgetting, payoff discounting, and procrastinating aspects of the model, showing that agent payoff discounting and lookahead predict the observed spike in Sunday night traffic, while forgetfulness can explain a decline in activity on Saturday where the utility of login should be increasing. We then use the learned parameters from this fitting to actuate agents visiting a social networking website hosted on a virtual machine, and we measure the impact of increasing or decreasing the perceived ease of login on the hourly volume of network traffic at peak times.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.2.11 [**Artificial Intelligence**]: Intelligent Agents

General Terms

Experimentation, Human Factors, Security

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

Keywords

Cognitive architecture; hyperbolic discounting; network traffic simulation; web actuation; software testing; virtual agents

1. INTRODUCTION

Network infrastructure and computer security have been acknowledged recently as becoming increasingly important to our thinking about national defense [9]. As both civilian and military critical infrastructures have become increasingly connected to personal computers and the Internet, preparing for cyberattacks has become a national priority. To this end, at several places in the United States, test environments such as the National Cyber Range have been created to host large-scale experiments for testing network defense measures [9]. These environments consist of server racks hosting a large number of virtual machines running real operating systems and applications, connected in configurations designed to mimic critical networks as closely as possible. These ranges provide a realistic but quarantined vulnerability testing environment for new software as well as a training environment for new network administrators.

Since the tests use real software and operating systems, the processes emulating the users who drive the applications are the limiting factor in achieving realism on these testbeds. The most common testing suite, LARIAT [17], has modeled users as simple augmented Markov models called "Petri Nets" [14]. These emulated users could perform some simple tasks on the machine, such as randomly switching between typing into word processing software and sending emails, but their control was essentially open-loop. While this approach does create a variety of incidental network traffic, the predictive power of such models would be sorely limited when attempting to model a real crisis in which user behavior patterns were significantly disrupted. Any tests of network defenses or training exercises conducted in such an environment would have no way of assessing the impact on network users attempting to achieve their normal goals on the network, nor any way of predicting natural changes in behavior that would result from their attempts to adapt. Robustness in a finite state machine depends on the foresight of the programmer, and the number of situations possible on a network makes hand-designing every response impractical.

The GOSMR architecture is an attempt to build realis-

tic agents that can perform goal-directed behavior in these test environments, creating a virtual community of "ghosts in the machine" operating the virtual machines on the network. GOSMR stands for "Goal-Directed Scenario Modeling Robots." If the user models in these environments act to pursue goals, then experiments in these environments can begin to model how user behavior changes and adapts to disruptive events on the network. The scenarios to model include the spread of malicious software, active cyberattacks and defensive countermeasures, events and deadlines in the physical world causing spikes in network activity, and the simple act of deploying new software on the network. Though no physical actuation is taking place, the agents ultimately must be "robots" in that they must perform sensing and actuation on their applications. This is a crucial distinction from a simulated environment in which real browsers and applications are not used: the agent must have some way of perceiving or knowing how to interact with webpages and applications, and an action that could be modeled in the abstract as a simple planning operator may actually consist of a series of clicks, mouse movements, and typed values. Dealing with real applications keeps the work honest in the same way that dealing with a physical robot keeps a roboticist honest, and new issues such as tabbed browsing, hidden text, and so on perpetually provide corner cases to address.

This paper is the first to describe the GOSMR architecture, but besides being an introductory architecture paper, the current work also evaluates a key assumption behind the architecture: *network behavior is not static, but is driven by users attempting to remember the past and predict payoffs in the future.* With Markov Models and Hidden Markov Models providing an extremely popular, easy, and effective means of generally modeling time series data [15], the Markov assumption of memorylessness has been supremely successful in a variety of arenas, and it was this assumption that drove earlier approaches to actuating network traffic [17]. But in dealing with data generated by real people carrying out their daily tasks on the network, it becomes apparent that they sometimes respond to events expected in the future, and that the amount of time remaining until those events occur can impact decisionmaking. The way in which human beings think about payoffs in the future has had an impact on psychology [11], economics[12], and the ACT-R cognitive architecture [7], but to our knowledge, has never before been used to explain activity on a computer network.

We argue that accurately modeling the ways in which human beings think about payoffs will have a large impact on the accuracy of models predicting not only network traffic patterns, but the likely impact and effectiveness of new security procedures. Individual users may not bear the cost of their bad decisions, as in the case of failing to keep machines clean and allowing them to become part of a zombie botnet [2]. Users can be overly tempted by immediate payoffs that bear uncertain long-term costs, such as following promising links of uncertain trustworthinesss. Security procedures that produce too high a burden on the user, such as preventing them from using software that is necessary for their job, may encourage users to circumvent the procedure altogether. It is through users' selfish or accidental deviations from expected behavior that vulnerabilities can creep in. Modeling how users react to changing incentives then becomes paramount.

The GOSMR architecture is designed to take parameters learned from data to build modular models of how incentives drive user activity, and then use these models to drive real applications such as web browsers. Each competing incentive is modeled as a Behavior that recommends a next action to take. These Behaviors are modular and composable, so that different software and network configurations can be tried without retraining or gathering new experiment data. The utilities of incentives can be trained from data in simulation, but then varied to ask how user activity might change if the incentives themselves were made more or less appealing. Action objects are hierarchical action descriptions that make it easy for a programmer to reuse code to actuate different browsers or perform different tasks. While it is a long-term goal to have action models learned by demonstration, we have concentrated here on making it easy for developers used to scripting approaches to build intuitive and easily testable models of actuation. We are currently focusing on the browser as the most common effector of network activity, but the issues faced in sending and receiving messages on a social networking site touch on many of the issues that would be necessary to actuate Office applications and email, as we currently plan. The architecture is also designed to be able to send human-readable, meaningful messages on the wire, so that human operators intercepting packets can understand their role in the scenario; and to be able to react gracefully to failures, by replanning around them. Neither is the focus of the present work, but we touch on these features because of this paper's role as the first GOSMR paper.

The main contributions of the present work are the following. We present for the first time the GOSMR architecture, a cognitive architecture which bears some resemblance to SOAR [10] and ACT-R [1], but which has been designed with an eye toward browser actuation and minimal computational overhead. We demonstrate the importance of modeling future discounting and forgetting in explaining network activity, by performing an experiment in which we train simulated versions of our models to produce appropriate distributions of Virtual Private Network (VPN) logins over the course of the weekend, including a spike that is best explained as the result of users both planning ahead and discounting future penalties. Finally, we demonstrate that we can take data gathered from real network activity, infer utilities and parameters of the user behaviors, and use these to drive agents that actually operate real websites, measuring the impact of the perceived difficulty of logging in on both real network traffic and a metric of user success in pursuing work-related goals.

2. RELATED WORK

The goals of the system are somewhat similar to those of the SOAR [10] and ACT-R [1] systems, two "cognitive architectures" designed to simulate human performance in real-world tasks. GOSMR's action choice system bears some similarity to these systems' rule-based approach. GOSMR directly borrows ACT-R's utility discounting function for future payoffs and its softmax choice function [7]. However, as rule-based production systems, ACT-R and SOAR would take significant work to deal with planning and reasoning over uncertainty, which are present and planned features of GOSMR, respectively. Some features of GOSMR exist in one previous architecture but not the other. SOAR has implemented communication, but ACT-R has not; on the other hand, SOAR is more deterministic in its execution

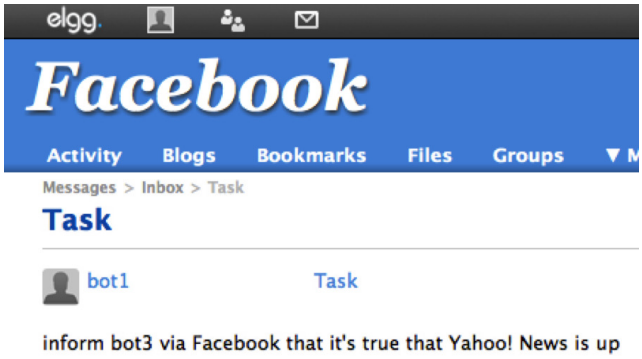

Figure 1: GOSMR is designed to capture goal-directed, coordinated behavior on the network while driving real websites and applications, such as this mock social networking site. The agents can instruct and inform each other using a recursive grammar that maps to agent beliefs and actions.

than ACT-R, which we feared may be a problem when attempting to produce diverse behavior on a network. The idea of modular behaviors recommending actions has also been used successfully in robotics in the DAMN architecture[16].

The idea of using a planner to solve computer security problems has been previously used in simulations of security problems, such as solving for how a nefarious user could gain access to an administrator's machine [5]. However, planners have not been used previously in large-scale network emulation experiments. The LARIAT architecture [17] is the standard for such experiments. It models users with Petri Nets [14], Markov-like models in which tokens progress asynchronously and probabilistically from one state to another, creating side effects as they progress. The GOSMR architecture is replacing these Petri Nets in the LARIAT architecture, and otherwise relies on LARIAT for issues such as pushing models to clients and experiment setup and teardown. Our partial order planner implementation closely follows [18].

The approach to sentence creation here closely resembles the Prolog approach to classic natural language processing [13].

A simple simulation of how user machine compromise would impact network administrator workload was discussed in [4]. Our assumption of power-law friend connections and the preferential attachment algorithm for producing them comes from [3].

Psychological research suggests that human thinking tends to be sloppy until effort is expended to make the decisions logical; Kahneman recently wrote an excellent popular review of the subject [11]. It has recently been proposed that human fallibility and short-sightedness may play an important role in modeling computer security problems [2]. The ability to remember planned tasks is called "prospective memory," and its role in pilot error is discussed in [6]. ACT-R also models the way goal productions are triggered by associative memory. Hyperbolic discounting as a psychological model is discussed in [7], while its role in explaining irrational economic behavior is discussed in [12].

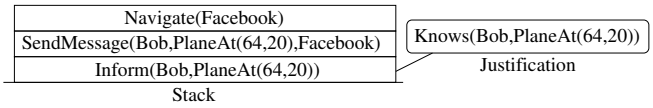

Figure 2: The agent uses a stack to carry out actions, allowing high-level actions to call low-level actions for code modularity and reusability. If an action fails, it may have a justification attached, allowing the partial-order planner to find a different action sequence that achieves this goal.

3. ARCHITECTURE

3.1 Thinking Fast and Slow: the Stack and Replanner

One goal for the architecture was to have the agents use a minimal amount of computational power to decide what to do next until they encountered problems, at which point they would be allowed to use more computation. We also wanted the agents' reasons for acting to generally be clear to simulation observers (and debuggers), such that someone just starting to observe an agent in the middle of a high-level action would still understand the reasons for its actions. Finally, we also wanted the agents to generally be very scriptable, so that programmers uncomfortable with planner -style action descriptions would be able to nevertheless easily create scripts at a high level. These goals drove the design of the stack-based architecture that agents normally use to perform actions (Figure 2).

When the agent's stack is empty, it is considered idle, and a Behavior module (see below) generates a new Action for the agent to pursue. An Action is an object that theoretically contains all the high-level information necessary to carry out a task – for example, if a SendMessage action is put on the stack, it should include the recipient and message text, as well as the website or application that will be used to send the message. In other words, all decisions have been made; all the agent needs to do is carry out this task. An Action object may put lower-level actions on the stack – for example, an InformMany action that tells many agents some information may put an InformAction on the stack, which can put a SendMessageAction on the stack, which may require a NavigateAction to get to a website where a message will be sent. Actions maintain the state information necessary to remember where they left off when a lower-level action finishes; thus, the core of the architecture is a simple hierarchical finite state machine (HFSM).

Action objects have act() methods that are called to perform the action in question, but in cases where the Agent is interacting with an external object, it may be that object's internal representation that informs the agent of the particulars. For example, an agent must have a SiteKnowledge object to efficiently interact with a website. The SiteKnowledge contains the map of links that leads from the website's homepage to a page that provides a particular functionality, or "affordance" [8], as well as the classnames to search for in the HTML to find relevant links, input fields, and information. Without prior knowledge of a site, the agent can hunt for relevant links and fields using a FindAffordance subaction, but it is not nearly as efficient.

When an action succeeds or fails, it pops off the stack

and informs the agent of this status. If an action failed, the agent checks whether it had a "justification" –a goal Belief that a Behavior or Action can optionally add when placing an action on the stack that explains why it is necessary. If it did, then the agent can attempt to achieve the same goal some other way. For example, the Inform action has the justification of *Knows(A,B)*, where A is the recipient and B is the Belief being communicated, and this could be achieved using a different website from the original site specified in the action. When attempting to repair an action, the agent calls a partial order planner [18], which generates a plan to achieve the action's justification. The new plan is then placed on the stack, with the justification attached to the action lowest on the stack.

When an action was not provided with a justification, actions are popped from the stack until a justification is reached. If no actions in the stack have a justification, the agent has no idea why it was doing what it was doing, it cannot repair the plan, and its Behaviors choose another action to pursue.

We use a dependency injection framework, Google Guice[1], to separate abstract representations of actions from the concrete Selenium calls that actuate the browser. A dependency injection framework allows the specific implementation of a class or method to be easily swapped out for another by changing a configuration module. Because experiments on a cyber range may involve a variety of different software configurations, separation of the details of actuation from the higher level description becomes important, especially for robust unit testing. Despite being very hierarchical, actions can be tested individually by using Guice to substitute dummy actions, SiteKnowledges, and other classes.

The combined architecture allows the agents' behavior to simultaneously adhere to externally imposed requirements on what specific actions to take, such as following a script for normal behavior or obeying a distribution gleaned from data through machine learning, while still performing rationally in completely novel circumstances.

3.2 Behaviors, Utilities, and Lookahead

A Behavior is an object that, when queried, produces a suggested action, a utility payoff for taking that action, and an expected time for the payoff. Some implemented examples include a SleepBehavior that recommends sleep but only produces a positive utility if it is past an agent's simulated bedtime; a NewsBloggingBehavior, which recommends going to a news website to find a news story, or posting a blog post about it if it already has one; and a CheckMessagesBehavior, which recommends checking one's messages on a website with utility proportional to the time since the last check. Because these behaviors provide their suggestions and utilities independently, Behaviors can be added or subtracted from an experiment independently. Behaviors may also be composed hierarchically – for example, a TimeDependentBehavior could recommend the output from one Behavior in the morning and the output from a different behavior after noon.

Some actions may have negative payoffs if they are *not* taken – for example, an agent may get in trouble on Monday if it has failed to check its email all weekend, as we model in Section 4. To preserve the independence of the Behaviors, we simply treat this as a positive utility for taking the action.

[1]http://code.google.com/p/google-guice/

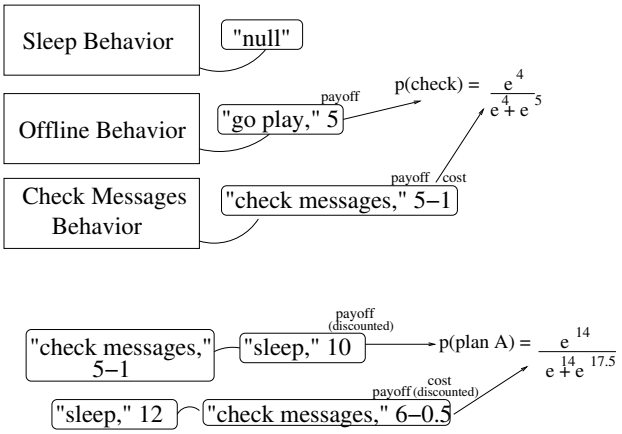

Figure 3: (above) Behaviors each provide a recommended action and a utility, and a softmax function decides which is chosen. (below) The behaviors are queried twice to create a two-step plan, with future costs and payoffs discounted, and then resampled to create an alternate plan. Costs seem more palatable as a second step while payoffs in the far future are unaffected, resulting in procrastination.

When an agent needs to know what to do next, the agent queries its Behaviors, and they recommend a two-step plan in the following manner. First, an action is selected from the Behaviors' recommendations according to a softmax function, $p(action) \propto e^{c*utility(action)}$, where c is a constant that determines how often the agent chooses suboptimally. Then, the agent selects the next high-level action that will be performed after that action, using the same softmax function, but discounting the payoffs and costs because it is occurring in the future (see below), given the estimated time it would take to perform the first action. (Subactions of the first action, such as navigating to a website in order to use it, are used to estimate the time of the first action, and do not count as the second action.) The agent then makes a second two-step plan using the same process. Having sampled two possible plans, the agent then uses a softmax function to choose between the two plans. It will only perform the first action of the chosen plan before resampling another two-step plan. Because of the way discounting makes both positive and negative payoffs in the future appear smaller, this leads to a tendency for the agents to create plans in which the bad payoff is performed second, and possibly perpetually defer that action. We chose to sample twice instead of comparing all possible two-step plans for efficiency. Figure 3 illustrates the process of choosing the next high-level action.

3.3 Time discounting

Human beings value future rewards and fear future penalties less than rewards and penalties in the present. Though exponential discounting is a more common means of discounting future rewards in artificial intelligence [18], human psychological research suggests that a hyperbolic discounting curve is more realistic for human behavior [7]. This is a curve described by the equation:

$$u(t) = \frac{u}{1 + kt} \tag{1}$$

$u(t)$ is the perceived utility of a payoff of u that is t units of time in the future, and k is a constant (that depends on the units of t). Hyperbolic discounting is different from exponential discounting in that an agent may have a different optimal choice at time t in the immediate future than at time t' in the distant future; the reward curves for two different sizes of rewards cross, resulting in a "smaller-sooner, larger-later" policy for agents deciding what to do. We believe this will be important in modeling security and network scenarios, in which short-term thinking can often lead to large losses later, but users know on some level what they ought to do for the long term (e.g., patch installation and anti-virus updates).

3.4 Forgetting

A user may not remember a task at the time a decision is being made. If a task is forgettable, we assume the probability that a user does not recall it is linearly proportional to the time between the present and either the time of the last reminder of the task, t_r, or the time remaining until the deadline, t_d: $p(forget) \propto min(t - t_r, t_d - t)$. The curve of the probability of remembering is therefore a V shape with endpoints at the last reminder and the deadline. The constant of proportionality that determines the slope of the V can be learned from data. Forgetting tasks is the only way to completely prohibit a Behavior from putting forward a task, since even actions of zero or negative utility otherwise have a small chance of being performed according to the softmax function. A forgotten task is not permanently forgotten – a Behavior rechecks for remembering the next time the stack is empty. Our assumption of linearity is for simplicity, and is not theoretically motivated.

3.5 Language

We chose to use grammars instead of n-gram models for language because the system is designed for measuring the impact of network disruption on normal user activity, and to do that, the messages need to convey real information that can be propagated through the network and affect agent actions. Moreover, we chose to make the messages human readable and purely in-band so that intercepted packet contents will be interpretable when the system is used in training exercises.

GOSMR agents use a simple recursive grammar that can parse or produce sentences that describe beliefs and commands. During production, agents use the grammar to write a string representation of a Belief or Action object (where Action strings are commands). During comprehension, the same grammar is used to extract the relevant fields from the sentence, and build the appropriate object. The grammar can be recursive – for example, a DelegateAction contains an Action as one of its fields, which is the action that should be delegated to another agent. The grammar indicates the meaningful strings to be extracted from the sentence and their types. Lookup tables convert these strings to Objects representing the things referred to, such as a SiteKnowledge object in the case of a named website. Java reflection is then invoked to construct the relevant Action or Belief with these arguments. Figure 1 shows an example sentence.

To speed up parsing, a prefix property is enforced on the grammar so that agents can simply look up which rule to use in constant time.

3.6 Offline behavior

User behavior offline can sometimes affect user behavior online; for example, if a user is asleep, the user cannot connect to the network. This means that we must in some instances model user behavior offline to accurately model online traffic levels. Luckily, the converse is also true: network behavior reflects user behavior offline, at least to the extent that it impacts the network. Fitting behavior models to network data, including offline behavior, is the subject of section 4.

4. EXPERIMENT 1: UTILITY DISCOUNTING AND FITTING TO NETWORK DATA

Evaluating the architecture in its entirety is beyond the scope of this paper; it is large, complex, and designed to be flexible to modeling needs. We instead report here an experiment designed to test the hypothesis that discounting future rewards has visible effects on network behavior. Our larger hypothesis is that many security-related behaviors, ranging from laziness in changing passwords to clicking on suspicious linkfs, can be informed by a general model that assumes people undervalue payoffs and penalties that occur in the future, and this is a first step toward that larger vision.

4.1 Methodology

As a natural network behavior dataset that reflects the way people trade off future payoffs and penalties versus present rewards, we used logs of Virtual Private Network logins over the weekend for an enterprise of 3121 users over the course of 59 weekends. We assumed that this behavior could be well-modeled by a combination of three behaviors: a behavior that provided a payoff for going to sleep after a particular time of night; a payoff for performing arbitrary offline actions over the weekend of exponentially distributed duration; and a payoff for VPNing in. In the network environment from which we get our data, VPN is normally used to check email, but it is rendered inconvenient by the security procedures surrounding login; thus we assume there is an immediate utility cost for logging in, but a greater cost at the beginning of the workday on Monday for not handling the messages.

We made the following simplifying assumptions in modeling the VPN decision task. We assumed there was some constant rate of incoming mail or other messages that a user would need to deal with over the weekend, each with some probability of having a bad payoff on 9AM the next work day if a VPN session did not occur. From the user's perspective, this leads to a constant expected negative payoff at that time. VPNing in can reduce this payoff by an amount proportional to the amount of time that had passed since the last login, modeling handling all the messages that have arrived since leaving work on Friday. If no utility discounting is occurring, we would expect the login rate to increase linearly over time during waking hours. If discounting is occurring, we would expect a spike in traffic just before the users go to bed on Sunday night.

We assume that users' exact bedtimes and the rate of incoming tasks are normally distributed, while the length of weekend activities is exponentially distributed. We also assume users sleep eight hours; any parameters varying this would presumably not be worth the complexity. This leads to the following model parameters: payoffs for sleep and

weekend activities; mean and variance for rate of VPN payoff; mean and variance for bedtime; expected duration of weekend activities; discount factor for future payoffs; noise factor for the softmax decision function; and the constant determining the linear relationship between time-from-reminder and probability of forgetting the task.

We adjusted for two other factors in the data set. First, three-day weekends shift when the negative payoff would occur for failing to handle tasks, so we shift the data on these weekends to be relative to the 9AM deadline, ignoring the first day of the three-day weekend and treating the other two days as the relevant data. Second, the users were scattered over 20 time zones, so we correct for this by looking up a user's latitude and longitude from their IP address of login, and correct their login trace to be relative to their own time zone's beginning and end of the weekend, so that we fit all user data according to user-local, deadline-relative time.

For each run of a set of agent parameters, we sampled 1 million runs of an agent from 8PM Friday to 7AM Monday to get the curve of expected logins by hour for the weekend. (We chose to avoid times around when users would travel to and from work, which would introduce additional parameters to model the reduced likelihood of login during travel.) Training was not actuated and occurred offline. For each condition, we sampled 100 random parameter sets, took the set with the lowest root mean square error compared to the training data as a starting point for hill climbing, and then performed hill climbing until convergence. Because the parameters had very different magnitudes, making gradient descent somewhat infeasible, our hill-climbing process consisted of choosing a random parameter and adjusting it up or down by a factor of ten percent. Convergence occurred within 200 runs in all cases.

4.2 Results

As Figure 4 shows, there is a spike in the logins on Sunday night, and this is the key place where the models differ in their predictions. Only a model with both hyperbolic discounting and modeled lookahead predicts the suddenness of the spike, resulting in the best fit to the weekend data ($R^2 = 0.92$). Exponential discounting predicts less traffic overall on Saturday ($R^2 = 0.81$), omitting lookahead from the model results in a failure to predict the full magnitude of the Sunday night spike ($R^2 = 0.71$), and omitting forgetting from the model results in a spike on Saturday night instead of an overall decrease in traffic ($R^2 = 0.68$). Omitting utility discounting altogether from the model (not shown) also results in a failure to predict the Sunday night spike ($R^2 = 0.76$).

Table 1 summarizes the results of Experiment 1, validating the importance of (a) modeling the way users can put off tasks by looking to the future, (b) modeling how users discount rewards and penalties in the future, resulting in an increased tendency to put things off until the last minute; (c) the importance of modeling forgetting, without which, the model cannot predict the sag in traffic over the weekend.

5. EXPERIMENT 2: IMPACT OF EASE OF LOGIN ON PEAK WEBSITE TRAFFIC

Having trained a model of how users value performing work-related communication over the weekend relative to other weekend activities, we can now answer some interesting questions about how actual network traffic would be impacted by changing the ease of use of a website. The scenario we wish to test is a proposed website internal to the organization's network that would become the new preferred way of users communicating over the weekend. We can assume that the parameters involved in the user activity are basically the same – we have not changed the value of sleep, nor how forgetful people are – except the cost of logging in, which will depend on how extensive the security procedures are surrounding the site. Making it available via the web, requiring only a login, would make the page more convenient than VPN, which requires using a security token. For more extensive security, the website could require both VPN and a login on the page. The convenience may impact how people procrastinate, which will impact both the severity of the Sunday night traffic spike and whether people complete their tasks at all. We examine the qualitative effects of making login easier or harder by examining the traffic patterns for half the login cost, the same login cost, and double the login cost of VPN.

Though Experiment 1 was performed in simulation, we demonstrate here using the Action and Behavior system described earlier to drive Chrome browsers visiting a real social networking website (Figure 1), powered by the Elgg social

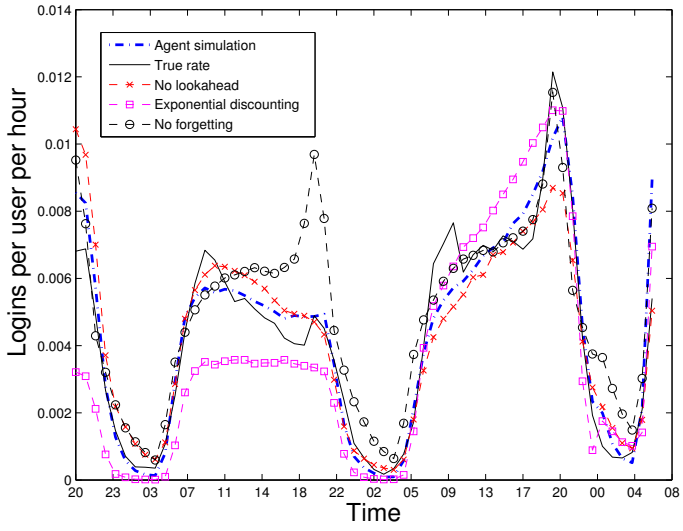

Figure 4: Fitting behavior parameters and utilities to the distribution of Virtual Private Network logins over a weekend. As utility discounting would predict, there is a spike in network traffic on Sunday night.

Model	R^2
Full model	0.92
Exponential discounting	0.81
No discounting	0.76
No next action lookahead	0.71
No forgetting	0.68

Table 1: Goodness-of-fit for login rates over the weekend using variations of the full model.

network framework and hosted on a virtual machine running Linux. By using tcpdump on the virtual machine's ethernet interface, we measured the hourly traffic, in packets and bytes, that would be generated by a small user group using the site over the course of the weekend. Thus, in addition to studying the effects of utility change, we also demonstrate that we can actually drive the site traffic, such that by running agents on many machines (we here just use one), we could perform realistic stress testing.

5.1 Methodology

Using the full model trained in Experiment 1 (lookahead, hyperbolic discounting, and forgetting), 25 GOSMR agents were deployed on Windows 7 desktop with a 2.4 GHz Xeon quadcore processor and 6GB of RAM. This machine also hosted a virtual machine running Ubuntu Linux, which hosted the website. The agents' behaviors and actions were the same as in the simulation except in the VPN or messaging behavior; instead of simulating login to a VPN server by waiting, the agents here navigated to the Elgg social networking website, logged in, navigated to the subpage where their messages were kept, read all unread messages by clicking on them one at a time, and if a message was read, the agents sent a message as well. Agents' message recipients were chosen randomly from among their contacts on the site; the contact network was created ahead of time according to a power law distribution, $P(X = x) \propto x^{-2}$, with a minimum value of 3. The agents all used the Selenium framework to actuate the sites, and used the Chrome web browser to perform their navigation. When an agent had run out of things to do on the site for a given session, it did not log out, but did close the browser – this was a simplification designed to enhance scaling, since copies of the Chrome browser itself were the limiting factor in consuming system resources.

The experiment was run at a 60x speedup, actuating the 64 hour full weekend from 5PM to 9AM in about an hour. The experiment began with agents possessing SendMessageActions on their stack before the weekend began with a fixed probability of 0.75, since we assume all message sending on the weekend would largely be triggered by messages left over from the work week; these messages were sent before the simulation began.

We measured the total hour-by-hour traffic in kilobytes for three conditions: a login that was exactly as inconvenient as the VPN, a login half as inconvenient, and a login twice as inconvenient. The experiment was repeated 10 times for each condition to assess whether differences were statistically significant.

5.2 Results

Table 2 shows the average and peak hourly bandwidth consumed by our 25 agents, as measured by tcpdump. Increasing the perceived cost of login significantly decreased the mean and peak traffic ($p = 0.0005$ and $p = 0.03$ by two-tailed t-test). While average traffic was more than halved, peak traffic decreased by only about a quarter, suggesting that much of the reduced traffic was simply deferred until the last moment. Otherwise, these results show that it is simply not worth the bother for many of our agents to log in to check their messages once the cost becomes high, and they simply choose to do weekend activities instead.

The work-related utility achieved by the agents, scored whenever they reacted to messages, was also significantly lower (5.7 vs. 1.1, $p = 0.03$). In fact, the work-related utility achieved was reduced by a greater amount proportionally than the overall traffic was reduced, since users were more likely to log in to find no messages, making the traffic useless. Hypothesizing a login process that was half as cumbersome produced no significant difference in mean ($p = 0.43$) or peak ($p = 0.69$) traffic. In the case of the doubled login cost, we interpret the large decrease in utility (which is more than halved) as the result of communication depending on both agents' cooperation, so the absence of either causes communication to fail. This is an example of an effect that can be explained in retrospect, but may have been difficult to anticipate, highlighting a need for simulation and experimentation.

Login cost	Mean Hourly kB	Peak Hour kB
Normal	240.7	1237.8
Half	211.2	1289.6
Double	104.0	975.7

Table 2: Peak hourly traffic to the website resulting from 25 agents who have the same, doubled, or halved utility cost of login compared to VPN, as measured by tcpdump, averaged over 10 runs. Increasing the perceived difficulty of login significantly reduces average and peak traffic.

6. DISCUSSION AND FUTURE WORK

Network traffic is driven by human users, and predictions of network traffic under unusual circumstances will only be as good as the models of the people driving the traffic. While training models to replicate observed action probabilities directly may be easier, there are good reasons in modeling network and security scenarios to prefer an explanatory model that seeks to get at the underlying utilities of actions. First, it allows us to more effectively ask "what if" questions about policy and procedures, modeling the way in which certain user activities will become more or less common if they are made easier or harder. Second, it provides us with a metric of the impact of disruptions to user activity – this metric being the lost utility compared to the normal baseline. Once we have trained a model to learn what is important and not important to the users, this weighted utility can now become a measure under different conditions of how important the disrupted activities were. When deciding to adopt one policy or another, or estimating whether a piece of software is negatively impacting user work performance, the estimated utility achieved by the users could be a better benchmark than sheer amount of traffic produced or sessions achieved because utility can vary by user and circumstance. In training scenarios for human operators, such metrics could be regarded as a rough "score" indicating how intrusive or effective their actions were.

The work here is a first step toward a more sophisticated model of how information travels through an enterprise network, largely because we are limited by the data we have available. We expect this to be a recurring problem in modeling network activity meaningfully; it is easy to install packet capture and get a rough idea of the volume of traffic passing between two machines, and rather more difficult to understand what purpose is being achieved with that activity, how important it is to the user, and how the

user might have reacted if the connection were severed. Despite the difficulty of answering these questions, that is our mandate; and thus, we are forced to find a way to make rule-based, expert-system-like, high-level descriptions of user activity meet in the middle with machine learning methods that can match the total volume over time that is observed. We would ideally like as much of the behavior to be driven from data as possible, but sometimes, the data is simply unavailable, and often, the probabilistic methods that are currently popular in machine learning do not produce behavior that makes sense. Our philosophy is to start with behavior that is meaningful; there will always be enough parameters to fit the data, and when the data cannot be fit, that is when something useful can be learned about the model.

The problem of realistic, scalable Web actuation is an excellent opportunity to integrate all kinds of methods that are rarely combined into whole agents. Dealing with real websites to actuate forces the agent designer to confront problems of attention, hierarchical organization, perception, and robustness that might be glossed over in a pure simulation. The problems we face of sensing, actuation, and robustness are therefore somewhat similar to the problems facing the roboticist; but the Web is a much more hospitable home for an artificial agent, since it operates at a more abstract level and in formats already conducive to machine processing. We therefore hope to make progress on the "whole agent problem" at a faster rate than the roboticist, who is usually mired in problems of vision and mechanics. Our future plans for the system include reinforcement learning, to learn the Action models for interacting with websites and applications automatically; probabilistic reasoning, so that the agents can mistrust the messages they receive and false claims coming from network intruders; and user topic modeling to model distributions of search engine queries and external Web traffic. We have also begun work on allowing the agents to interact with a simulation of the physical world so that we can model the ways in which information about events transpiring in one location can quickly spread to other locations.

Given the way in which user activity spikes before a deadline, accurately modeling user behavior is essential to understanding the stresses that will face the network in a real crisis. We have shown that discounting of future payoffs and the users' ability to defer tasks combine to form a last-minute spike, and changing the payoffs facing a user can reduce or increase the intensity of that spike. We hope in future work to also show that the manipulation of payoffs can have an impact on security-related scenarios as well.

Acknowledgements

The authors gratefully acknowledge Lee Rossey, Marc Zissman, and Bernadette Johnson in supporting and promoting this work; the LRNOC team, especially Kim Hebert and Vernon Rivet, for providing the infrastructure from which the data of Experiment 1 was procured; and Tim Braje, Sarah Chmielewski, and Michelle Sternowski of the LARIAT team for their work in integrating GOSMR with LARIAT.

This work is sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

7. ADDITIONAL AUTHORS

Additional authors: Mark Mazumder (MIT Lincoln Laboratory, email: `mazumder@ll.mit.edu`)

8. REFERENCES

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4), 2004.

[2] R. Anderson and T. Moore. Information security: where computer science, economics and psychology meet. *Phil. Trans. R. Soc. A*, 367:2717–2727, 2009.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439), 1999.

[4] J. Blythe, A. Botello, J. Sutton, D. Mazzocco, J. Lin, M. Spraragen, and M. Zyda. Testing cyber security with simulated humans. In *Proceedings of IAAI*. AAAI Press, 2011.

[5] M. Boddy, J. Gohde, T. Haigh, and S. Harp. Course of action generation for cyber security using classical planning. In *Proceedings of ICAPS*. AAAI Press, 2005.

[6] K. Dismukes and J. Nowinski. Prospective memory, concurrent task management, and pilot error. In A. Kramer, D. Wiegmann, and A. Kirlik, editors, *Attention: From Theory to Practice*. Oxford UP, 2007.

[7] W.-T. Fu and J. R. Anderson. From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General*, 135(2), 2006.

[8] J. J. Gibson. The theory of affordances. In R. Shaw and J. Bransford, editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*. Lawrence Erlbaum, 1977.

[9] W. J. L. III. Defending a new domain: The Pentagon's cyberstrategy. *Foreign Affairs*, 89(5), 2010.

[10] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. Kenny, and F. V. Koss. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1), 1999.

[11] D. Kahneman. *Thinking Fast and Slow*. Farrar, Straus, and Giroux, 2011.

[12] D. Laibson. Life-cycle consumption and hyperbolic discount functions. *European Economic Review*, 42:861–871, 1998.

[13] F. C. Pereira and S. M. Shieber. *Prolog and Natural Language Analysis*. Microtome Publishing, Brookline,MA, 1987/2002.

[14] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Upper Saddle River, NJ, 1981.

[15] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 1989.

[16] J. Rosenblatt. DAMN: a distributed architecture for navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3), 1997.

[17] L. M. Rossey, R. K. Cunningham, D. J. Fried, J. C. Rabek, R. P. Lippmann, J. W. Haines, and M. A. Zissman. LARIAT: Lincoln adaptable real-time information assurance testbed. In *Aerospace Conference Proceedings*. IEEE, 2002.

[18] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 2003.