Performance Analysis of Real Traffic Carried with Encrypted Cover Flows

Nabil Schear
University of Illinois at Urbana-Champaign
Department of Computer Science
201 N Goodwin Ave, Urbana, IL USA
nschear2@uiuc.edu

David M. Nicol
University of Illinois at Urbana-Champaign
Information Trust Institute
1308 West Main St. Urbana, IL USA
dmnicol@uiuc.edu

Abstract

Encrypted protocols, such as SSL, are becoming more prevalent because of the growing use of e-commerce, anonymity services, and secure authentication. Likewise, traffic analysis is becoming more common because it is often the only way to analyze these protocols. Though there are many valid uses for traffic analysis (such as network policy enforcement and intrusion detection), it can also be used to maliciously compromise the secrecy or privacy of a user. While the payload can be strongly protected by encryption, analysis of traffic patterns can yield information about the type and nature of traffic. In this paper we use simulation and an analytic model to examine the impact on user experience of a scheme that masks the behavior of real traffic by embedding it in synthetic, encrypted, cover traffic. This point provides a novel context where we observe the synergy of simulation and analytic modeling. We show that a detailed simulation model of network traffic characteristics can be used to estimate the parameters of an analytic model of tunneling.

1. Introduction

Network encryption, both at the packet and session layer, is used widely for securing private data. VPNs and encrypted sessions which use SSL provide a high level of message secrecy and integrity. SSL does not however, block an adversary from analyzing the behavior of the traffic for clues. Though seemingly innocuous, these clues can often result in a breach of secrecy for the user.

To further motivate this problem, consider the following example of HTTPS traffic analysis. The semantics of Web browsing is commonly that a small request is followed by a larger response. This behavior is not masked by SSL. An adversary (Alice) with appropriate access can see the encrypted payload as it crosses the network. She can also see the boundaries between requests and responses. Therefore, if Alice has access to the encrypted content on a site (for example, she and the victim (Bob) both have on-line banking accounts at the same bank) then Alice can catalog the size of each piece of content on the site. Once she has done this, she can snoop Bob's traffic and reliably determine the pages he has visited. Alice can deduce, for example, that Bob transferred money because he has been to the *transfer-funds* and the *transfer-completed* pages.

SSL traffic analysis attacks have been the subject of academic and security research since SSL was introduced [8]. Numerous attacks have been proposed and demonstrated which range from being able to detect the protocol of an encrypted VPN connection to extracting passwords from SSH encrypted sessions [2, 9, 1, 7, 5, 6]. Many of these attacks are surprisingly accurate and effective, despite the heightened sense of security which strong encryption provides.

The secrecy lost by these attacks varies depending on the application and the needs of the user. Realistically, many common uses of SSL do not warrant extra effort to prevent traffic analysis. However, a growing number of applications, (such as low-latency anonymity systems) and users, (such as privacy advocates) need better protection from these attacks.

Most SSL traffic analysis attacks capitalize on extracting information from behavior or protocol semantics. SSL does not mask timing behavior, boundaries, connection utilization, or PDU sizes. One simple approach to preventing this type of analysis is to normalize these behaviors (i.e., use fixed message transmission times, PDUs, etc...). But fixed patterns are themselves a clue that the traffic may be interesting. An alternative approach is to hide the fact that the traffic's behavior is being obscured at all. The idea is to have the system steadily generate *cover traffic*, which mimics the behavior of a hypothetical user. Users visit various websites, with queries that vary in size from site to site. They also utilize various encrypted services like SSH, Skype, or HTTPS. The URLs, queries, and data of the real, hidden users must be entirely contained within the mimicked traffic, which impacts the latency and throughput of the real traffic. All of the

mimic traffic that is **not** real traffic is overhead for the technique.

Our aim in this paper is to evaluate what effect such tunneling has on the real traffic being sent. Specifically, what happens when the tunnel traffic and real traffic differ greatly? How would the throughput of the real traffic be affected by various tunnel traffic? And most importantly, what degree of interactivity is lost when tunneling over similar or disparate traffic models?

We developed a simulation model of this type of scenario using SSFNet [3]. The model has detail with respect to protocols running in the hosts, but is simple in its representation of the network. This paper describes the model, and preliminary results analyzing the effects of tunneling. It then develops an analytic model that characterizes performance as a function of traffic characteristics, and validates the model's predictions against measurements observed in the simulator. Using the validated model, we develop expressions for slowdown due to tunneling, and consider attributes of cover traffic that are necessary to avoid loosing real traffic.

2. Simulation Model

We model all tunnel and real traffic as request/response pairs. Each request/response pair is a single *flow*. This model captures the behavior of many protocols and provides a consistent means to specify both model and real traffic. The size of requests and responses can be read from a trace file, or be generated randomly from any one of a variety of probability distributions.

There are two hosts within the system: a client and server. Each generates cover traffic. Conceptually, the client is the host which originates the cover connection and issues the request. The client host for the cover traffic is static and may not be changed; unlike the client for the real traffic, as explained later. The cover traffic server sends a response to the client after receiving its fake request. As we are interested in impacts on real traffic that are much larger in magnitude than network latencies, we need not (and do not) model the communication network between host and client with any significant detail. Furthermore, we will see that the traffic itself has high variance, which implies that the behavior already has significant effects we would expect to see using a more detailed network model.

The core simulation loop continually plays back the cover traffic flows over a simulated TCP connection. A cover traffic flow is characterized by a request of specified size, followed immediately by a response by the server. After each connection completes, the simulation pauses for a specified period of time and then begins a new cover flow. The delay and throughput of these flows are reported to the user. We obtain baseline performance of the real traffic by measurement we take using only cover traffic flows. Later we can compare the baseline against the performance of the same traffic embedded in a tunnel.

While the cover flows are being generated, available real traffic can be passed over the tunnel. Either the cover flow client or server may be the client for the real traffic. The real traffic client host is called the master and the server is called the slave. The real traffic master need not correspond with the cover flow client. The master generates a request a specified amount of time into the simulation. This request is timestamped at creation to calculate its total delay. The request is queued until the cover flow next transmits data. This real traffic request can be embedded in either a request or response of the cover traffic. The system determines how much space is available in the cover traffic transmission and consumes the correct amount of data from the real queued request. This process continues until the request is fully sent (potentially across multiple distinct cover flows). Once the slave receives the request it waits a specified amount of time and then generates and sends a response in a similar fashion. At the end of the simulation, the master and slave each report the throughput of the real traffic and the overall message delay for the response and request respectively.

3. Simulator Implementation

We have modeled this system using SSFNet [3]. The model contains two protocol hosts which use blocking sockets running TCP: blocking_tcp_server and blocking_tcp_client. We adapted the TCP server and client to play back cover flows automatically, and also play back generated model traffic if so specified in the model configuration file. We can specify distributions and moments for request and response sizes which the server and client will use . Another configuration parameter specifies how long the client waits between cover flows.

The client communicates certain information to the server to govern the generation of the simulated message sizes. Each message contains the sending time, the size of the request, the size of the response, the type of the message, and the number of bytes from the real traffic are being transmitted by this tunnel message. Our simulator models an ideal tunnel implementation where space consumed by this per-message overhead does not reduce the amount of space available in a cover message for real traffic. The client chooses a request and response size, and the server interprets both from the message. We use the send time in the message to compute the message delay once all the bytes are received at the other host. We use the message type attribute to selectively encode real traffic in the tunnel.

The configuration for the real traffic may be specified to either the cover flow client or server using the real_traffic list-attribute. The mode attribute (master, slave, no-model)

sets the role of the host. No-model specifies that only cover traffic be generated, which is useful for establishing a performance baseline of the network model. If left unspecified, the mode automatically defaults to slave. When mode is set to master, the user must also specify the distribution and moments of the real request and response size distributions. There are also two optional parameters which specify the timing of the real traffic. One specifies how long into the simulation the real request should start, while the other defines how long the slave waits after receiving the request before generating the response.

Both client and server contain a timer class to implement the request lead time or response delay (depending on which is the master). When fired, the timer's code generates either a request or a response, as appropriate. Since each simulation will only show the effect of a single real flow, the model needs only to record the aggregate amount of requested traffic to send, and the aggregate amount of traffic that has been received, to capture queued real messages awaiting enough tunnel traffic to be communicated.

Each message also has a state code which specifies the connection state (new, empty, and continue). The empty state simply tells the server to ignore the message because no real traffic needs to be transmitted. The new state instructs the slave that a new real flow is starting and gives the parameters for its behavior. The continue state encodes the number of real bytes sent in an existing real flow. Both master and slave keep track of the number of bytes remaining to be received on a real flow. When receiving part of a real message, the slave determines whether the message is a request or a response by a flag in the message called initiator. It then decrements the local state of how many bytes remain to finish the message. If the message has been completed, it will print a report on the overall throughput of the system and then generate a response. Minor bookkeeping is needed to support computation of real traffic bandwidth and latency.

4. Evaluation

Our evaluation uses a simple network that connects two routers with a 50ms delay and 100Mbits/s bandwidth link; each router has one attached host (client, and server); host to router links have 20ms delay and 1.5Mbits/s bandwidth. While simple, this network provides a useful baseline to quantify the effects of tunneling on a normal Internet user. We ran each experiment 10 times with the mean and standard deviation shown for each. Each iteration, the random number generator seed was set to a distinct value.

	request	*text*	*graphics*	*heavy*
mean	4456	17638	31347	49085
std dev	1709	2338	9970	13155

Table 1. Baseline Throughput (bytes/sec).

	request	*text*	*graphics*	*heavy*
mean	9.267e-2	0.586	0.813	1.053
std dev	8.792e-4	9.495e-2	0.1550	0.162

Table 2. Baseline Delay (sec).

4.1. Disparate Response Traffic Patterns

We explored the behavior of the HTTPS traffic because it is the dominant encrypted protocol used on the Internet today. As stated, HTTPS is (generally) a small request followed by a larger response. We instrumented local HTTPS usage and observed that the mean size of an HTTPS request issued from Mozilla Firefox is approximately 500 bytes, with variance 150. We measured response size as well and found it to vary greatly depending on the nature of the content being browsed. Thus, we identified three classes of response size distributions: Text (mean 10KB, variance 2KB), Graphics (mean 30KB, variance 10KB), and Heavy (mean 60KB, variance 20KB). While these parameters do not fully describe Web browsing, they establish reasonable categories to evaluate disparities between different types of traffic. Thus, we can evaluate the performance resulting from different mixtures of these categories for the tunnel and real traffic. Low observed variance in the request sizes suggest we use only one size category for requests.

We ran the baseline no-model tests with the parameters from these categories on the network configuration specified above and show the results in Table 1 and Table 2.

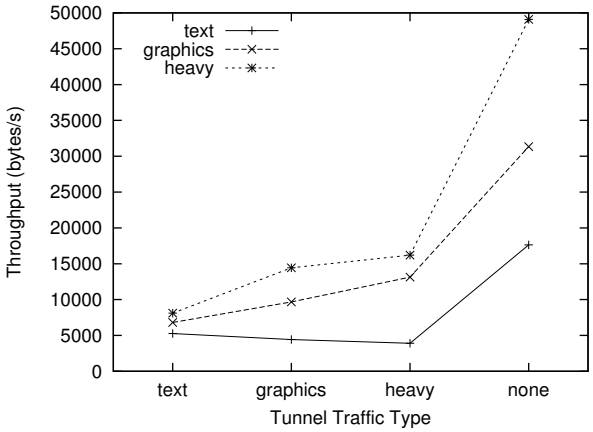
To evaluate the effects of tunneling, we layered one traffic model on top of another. To denote each tunnel test, we specify a traffic model which runs the real traffic first followed by a slash and then the tunnel traffic model. For example, we call a test running graphics traffic responses over text responses: graphics/text.

The first test we performed considers the effects of tunneling a real protocol over a tunnel which uses the same traffic model. The canonical example is running an HTTPS client model on top of another HTTPS client model with varying traffic properties. The results are shown in Figure 1. Each real traffic type is shown by a series on the graph and the tunnel traffic type is indicated on the x axis. We found for graphics and heavy real traffic that throughput increased with increasingly higher bandwidth tunnel models. Real text traffic tends to decrease in throughput with larger tunnel models. This counter intuitive result is due to the way

our simulator (modeled on a real implementation) processes messages and received data. The implementation does not deliver the real traffic until it receives the entire message of the tunnel traffic which carried it. Since the text model data is small, the tunnel message size dominates resulting in added delay and lower overall throughput. We observed that the bandwidth of real text traffic to be more consistent if message data was counted as soon as it arrived.

Overall, the best throughput for text, graphics, and heavy running over another HTTPS client model is approximately 64-70% worse than the same traffic sent without tunneling. In the worst case, heavy/text, the throughput was 83.5% worse.

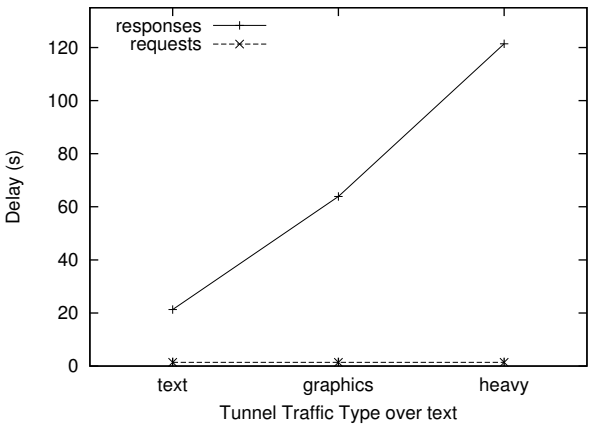
Figure 1. Throughput of HTTPS responses where both the real traffic and cover traffic are using HTTPS client traffic models.



A more interesting test is how the real traffic would perform if it was run on the reverse traffic model. Since the HTTPS traffic model is not symmetric, this will result in considerable effects for the real traffic responses. Each time a cover flow client requests a file, the real traffic response is sent. We tried each of the three response categories across a tunnel model that ran the request model (500 bytes variance 150 bytes).

The throughput between requests and responses were nearly identical because they are both use the same size cover traffic message (a single request) to transmit data. Delay remains static for requests because it only takes a single response from any of the categories to transmit the request. The effects on response delay are much more substantial. For this model, the throughput for responses has been reduced by up to 2 orders of magnitude. This is also expected as it takes many round trip times and many connections to transmit the entire message. For example, for a 10k message, it would take *at least* 20 cover flows to transmit the message.

Figure 2. Delay of HTTPS responses where cover traffic is an HTTPS client model and the real traffic is the text category HTTPS responses.



5. Analytic Model

A simple analytic model allows us to explore the interrelationships between model parameters and how the characteristics of the cover flow and real flow interact to affect the performance of the real flow when tunneled. We develop the model, validate it, and use it to derive bounds on the performance of tunneled traffic, and to describe constraints on the cover flow that are necessary to ensure that the throughput of real traffic when tunneled is the same as that when it is not.

5.1. Model Details

We suppose that the data sizes of both cover flow and real flow sessions are measured in a common unit, say kilobytes, and that with respect to these units, the size of a session is a random number of units, geometrically distributed. We describe these random variables G_c with success parameter p_c for the cover flow, and G_r with success parameter p_r for the real flow; the number of units in a cover flow session has mean $1/p_c$, and the mean number in a real session is $1/p_r$. The time required to transmit a kilobyte of the cover flow is denoted τ_c, and the corresponding mean for a real flow kilobyte sent without tunneling is τ_r. These constants incorporate all the effects of the network on packet transmission, e.g., TCP congestion control, packet loss, and link bandwidths.

Tunneled real flow performance depends on the delays between real sessions, and the delays between cover sessions. We assume that these delays are random, independent, and exponentially distributed. The mean cover flow

inter-session delay is μ_c, while that for real flows is denoted μ_r.

We can view the process of a cover flow as an alternating "on-off" renewal process [4], where the on time has the distribution of $\tau_c \cdot G_c$, and the off time is exponential with mean μ_c. We can also view the process of a real flow as an on-off process. The off time distribution is exponential with mean μ_r; the on time distribution is a bit more complex; we call a random variable with that distribution O_r. Finding the mean of O_r is the key to determining throughput and delay.

O_r has two components. The first is the time between when the real flow session begins, and when the next cover session that carries it begins. The mechanics of the implementation we model require that the real flow session wait until a new cover flow session begins. This first component may be time waiting for a cover session to complete plus a full inter-session delay, or it may be the remaining time in the current inter-cover session delay. Conditioned on knowing that the real flow starts while a cover session is active, the mean delay is the mean residual cover session time (τ_c/p_c by the memoryless property) plus μ_c; conditioned on the cover session being inactive, the mean delay is μ_c, again by the memoryless property. According to the theory of alternating renewal processes, the probability of finding the cover flow in the on state is $(\tau/p_c)/(\mu_c + \tau/p_c)$. Therefore the mean time between when the real flow starts and when the first cover session that carries it begins is

$$\left(\frac{\tau_c/p_c}{\mu_c + \tau_c/p_c}\right)(\tau_c/p_c) + \mu_c.$$

The second component begins with the start of the first cover flow session that carries the real flow, and ends with the termination of the last cover flow to carry it. The assumption of geometric distributions for the session sizes simplifies the analysis (for instance, that a real flow requires n kilobytes). Once a cover flow session begins to carry it, with the completion of every cover flow kilobyte there is a probability p_c that the flow session ends. Thus, at the end of each of the first $n-1$ kilobytes in the real flow, with probability p_c there will be an interruption and a wait for a new cover flow session. This implies that conditioned on a real flow session size of n, the number of waiting periods between cover flow sessions that carry it is a Binomial $B(n-1, p_c)$ random variable, with mean $(n-1)p_c$. The session that carries the last kilobyte of real flow might not terminate at the same time as the real flow. Due to the memoryless property, the remaining number of cover session kilobytes after the real session ends is geometrically distributed with parameter p_c. Thus, conditioned on n real flow kilobytes, the mean time between when the real flow is first carried and when it is considered to be finished is $n\tau_c + (n-1)p_c\mu_c + \tau_c/p_c$. The unconditional mean time to carry a real flow session is thus

$$\begin{aligned} E[O_r] &= \left(\frac{\tau/p_c}{\mu_c + \tau/p_c}\right)(\tau/p_c) + \mu_c \\ &\quad + \sum_{n=1}^{\infty}(1-p_r)^{n-1}p_r\Big(n\tau_c \\ &\quad + (n-1)p_c\mu_c + \tau_c/p_c\Big) \\ &= \left(\frac{\tau/p_c}{\mu_c + \tau/p_c}\right)(\tau/p_c) + \mu_c \qquad (1) \\ &\quad + (1/p_r)(\tau_c + p_c\mu_c) - p_c\mu_c + \tau_c/p_c. \end{aligned}$$

Delay is the time between when the first kilobyte of a real session is available and when the last kilobyte of that session is finally delivered. The mean delay is identically $E[O_r]$.

5.2. Validation

Before we use the analytic model to explore system behavior, we consider first how well it describes the simulation data we have collected already.

The data reported in Tables 1 and 2 are from models where the mean data loads as a function of traffic type are 10KB (text), 30KB (graphics), and 60KB (heavy). We chose parameters for the geometric distribution to match these means. Specifically, we use 1/10, 1/30, and 1/60 as the probability parameters of the geometric session size random variables. The variance of geometric random variables is considerably larger the variance reported in those tables.

We use Table 1 to get values of τ_c and τ_r for our model. Recall that these constants capture all networking effects on that transmission. While realistic values of τ_c and τ_r require detailed simulation, for the purposes of model validation we don't need to **predict** them. Given a throughput from Table 1 for a traffic type, when using that type as a cover flow, we take τ_c to be the inverse. So, for a text cover flow we take $\tau_c = 1/17.6$ seconds/kilobyte, for graphics we take $\tau_c = 1/31.3$ seconds per kilobyte, and for heavy we take $\tau_c = 1/49.1$ seconds / kilobyte.

The simulation study used a constant 200 msec delay between successive cover sessions; we assume the distribution is exponential with mean 200 msec. The model allows for delays between successive real traffic sessions, but this quantity does not factor into the delay or throughput equations.

To validate the analytic model, we compute the relative error of the predicted delay. Specifically, for a given mix of cover and real traffic type, we let d_{obs} be the mean observed delay from our simulation , let d_{pred} be the mean delay predicted by equation 1, and compute the relative error as $(d_{obs} - d_{pred})/d_{obs}$. The result is presented in Table 3. For

real / cover	text	graphics	heavy
text	0.03	−0.04	−0.02
graphics	0.18	0.06	−0.17
heavy	0.22	0.02	−0.04

Table 3. Relative error of predicted delay.

the purposes of using the model parameters to understand how their interactions affect performance, this agreement is quite good.

So we see that with accurate values of τ_r and τ_c the simple analytic model can predict performance reasonably well. This highlights one of the main points we want to make, that with our simulation plus analysis approach, the whole is greater than the sum of the parts. The traffic used to estimate these parameters need have nothing to do with tunneling. It can come from known traces, or it can come from detailed simulations that capture the effects of new transport protocols or that focus on behavior of a network under attack (e.g., denial of service). Traffic analyses that are ignorant of tunneling can tell you nothing about tunneling. Owing to the sensitivity of our model predictions on unknowns τ_r and τ_c, our analytic model without accurate measures of network delays gives predictions that are necessarily qualified. Together these components can be used to analyze the performance of tunneled traffic, as a function of traffic characteristics.

5.3. Analysis

Next we use the analytic model to better understand how characteristics of cover and real flows may affect performance of the tunneled flow.

5.4. Slowdown

Our first consideration is of the impact tunneling has on real traffic delay, as a function of the session size parameters p_c and p_r. It is intuitive that the mean delay $E[O_r]$ should increase as the mean cover session off-time μ_c increases. This is borne out rigorously by computing the derivative of $E[O_r]$ with respect to μ_c.

$$\begin{aligned}\frac{d}{d\mu_c}E[O_r] &= -\left(\frac{\tau_c}{p_r}\right)^2(\tau_c/p_c+\mu_c)^{-2}+1 \\ &\quad +p_c(1/p_r-1) \\ &= -\left(\frac{\tau_c/p_c}{\tau_c/p_c+\mu_c}\right)^2+1+p_c(1/p_r-1) \\ &> 0.\end{aligned}$$

The inequality follows because the negated squared fraction is always less than 1, and $(1/p_r - 1)$ is always positive. In fact, taking the 2^{nd} derivative with respect to μ_c, we obtain a positive value, which says that $E[O_r]$ is increasing and convex in μ_c. This tells us that the larger μ_c is, the larger $E[O_r]$ becomes, faster. It follows then that all other things being equal, delay is minimal when $\mu_c = 0$.

Now we compute *slowdown*, the ratio of the time required to deliver a tunneled real flow divided by the time required natively by that same real flow. To compute slowdown it is important to compute this ratio assuming exactly the same number of real flow kilobytes delivered in both schemes. Condition on the real flow having n kilobytes. Allowing for randomness still in the the cover flow behavior, we use Equation 1 to compute the mean delay for the tunneled session. The ratio of this mean to the time needed natively to deliver n real kilobytes is

$$\left(\frac{\tau_c}{\tau_r}\right)\left(\frac{2}{p_c n}+1\right).$$

It is worthwhile pausing at this point to observe just how critical it is to get the value of (τ_c/τ_r) correct if we are *predicting* slowdown. This ratio appears as a factor in the slowdown equation for every n, and as n grows the contribution to slowdown of the other expression diminishes. We comment more on this in the section on Validation. To obtain the mean minimal slowdown, we take the expectation of this with respect to the distribution of real kilobytes. As n appears in the denominator of the expression, this computation has no easily expressed solution. However, we know by Jensen's Inequality [4] that if $g(n)$ is a convex function (note that $g(n) = 1/n$ is convex), then $E[g(n)] \geq g(E[n])$. This means that a *lower bound* on the expected slowdown is obtained by replacing $1/n$ in the expression above by p_r, which leads to the proposition below.

Proposition 1 *If the cover flow and real flow sessions have sizes that are geometrically distributed with parameters p_c and p_r respectively, then the expected slowdown of real traffic due to tunneling is at least*

$$\left(\frac{\tau_c}{\tau_r}\right)\left(2\frac{p_r}{p_c}+1\right).$$

The ratio τ_c/τ_r above is intuitive, as it describes a performance difference observed with every transmission. τ_r and τ_c should not be thought of as independent from p_r and p_c though, for the large differences we see in observed throughputs (and hence values of τ) in Table 1 are due to the size of the sessions and the behavior of TCP congestion control. The smaller p_r and p_c become, the larger τ_r and τ_c become. There are limits though to variation in τ due to TCP mechanisms. Assuming loss-free operation, the maximum average transmission time is obtained when the flow

consists of one packet; the minimum average transmission time is derived from the link bandwidth.

The slow-down expression helps us understand the factors that dominate performance at extremes. When real flow sessions tend to be much longer than cover flow sessions (i.e., $p_r \ll p_c$) then the term p_r/p_c is close to zero, the extra waiting the real flow does at the beginning and end are insignificant compared with the time when the flow is actually tunneling, and the slow-down is overwhelmingly due to τ_c/τ_r. This ratio is greater than 1, reflecting the factor that the real flow would enjoy better average transmission times when TCP is reacting to the real flow session, but is paying a cost for tunneling through a choppier flow. Note that (τ_c/τ_r) is bounded; the performance hit cannot get arbitrarily large as p_r gets arbitrarily small.

By contrast consider the case where $p_c \ll p_r$. Now the ratio τ_c/τ_r is less than one—but is bounded from below. However, the term $(2p_r/p_c + 1)$ gets arbitrarily large as p_c grows arbitrarily small. This says that the slowdown is dominated by the extra time the real flow waits for a cover session to begin, and for the final one that carries it to end.

It is also natural to consider when the cover and real flows are described by identical parameters, i.e., $p_r = p_c$. The ratio $(\tau_c/\tau_r) = 1$, and so the expected slowdown is at least 3. This is intuitive, as we recall that the session lengths are assumed to be geometric. The time the real flow spends waiting at the beginning has the same distribution as the time spent actually carrying the flow, which has the same distribution as the time spent waiting for the final cover flow session to end.

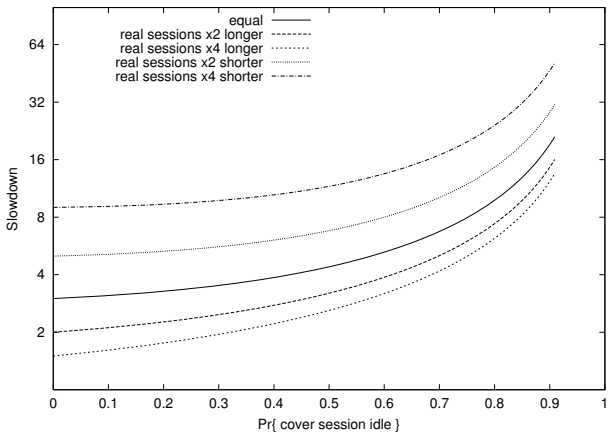
Of course, slowdown is larger in all these cases when $\mu_c > 0$. If we repeat the derivation that leads to Proposition 1, but carry along μ_c, we obtain

Proposition 2 *If the cover flow and real flow sessions have sizes that are geometrically distributed with parameters p_c and p_r respectively, and $\mu_c > 0$, then the expected slowdown of real traffic due to tunneling is at least*

$$\left(\frac{\tau_c}{\tau_r}\frac{p_r}{p_c}\right)\left(\frac{\tau_c/p_c}{\tau_c/p_c+\mu_c}+1\right)+\frac{\mu_c}{\tau_r}(p_r-p_rp_c+p_c)+\tau_c/\tau_r.$$

We now use this expression to explore the impact μ_c has on slowdown. To reduce the size of the parameter space to explore, we assume that $\tau_c = \tau_r$, in units where the common value is 1. Then we consider how slowdown varies as a function of μ_c, for a fixed set of values for p_c and p_r. We increase μ_c from an initial value of 0 up to a value for which the probability of finding the cover flow in the off stage is 0.95. Using a baseline of $p_c = 0.1$, we illustrate in Figure 3 how the lower bound on slowdown behaves when $p_r = 4pc_c$ (labeled "real sessions x4 shorter"), $2pc_c$ (labeled "real sessions x2 shorter), p_c (labeled "equal"), $p_c/2$ ("real sessions x2 longer") and $p_c/4$ ("real sessions x4 longer"). The independent variable is the fraction of cover session off time as the independent variable—which increases as μ_c increases—and the dependent variable is the lower bound on slowdown. Here we see a clear ordering of relative performance based on whether the real flow or cover flow is choppier (with preference to the choppier cover flow), with a dramatic—even alarming—degradation in relative performance as the cover session becomes increasing idle.

Figure 3. Slowdown as a function of cover session idle time.



5.5. Stability

We have seen that tunneling has the potential for slowing down the delivery of real traffic. If that slow-down is too large, processing of real traffic will not be able to keep pace with the generation of native real traffic (assuming here a decoupling of traffic generation from response). An important question asks after the characteristics of the cover flow that are necessary to "keep up" with a real flow.

We approach the problem by thinking of the tunneling as a service, and the real sessions to be tunneled as jobs to be served. We then have a queueing system with one server, whose mean service time is $E[O_r]$. The inter-arrival time between real sessions is the time between the beginning of real sessions when they are run natively. The mean of this inter-arrival time is $\tau_r/p_r + \mu_r$, where μ_r is the mean "down" time between successive delivered real sessions, run natively.

We know from the theory of $G/G/1$ queueing systems [4] that the queue is stable (i.e., has a limiting state occupancy distribution) when the mean service time is strictly

smaller than the mean inter-arrival time. In our case that is

$$\begin{aligned}\left(\frac{\tau/p_c}{\mu_c+\tau/p_c}\right)(\tau/p_c) &+ \mu_c + (1/p_r)(\tau_c + p_c\mu_c) \\ &- p_c\mu_c + \tau_c/p_c \\ &< \tau_r/p_r + \mu_r.\end{aligned}$$

There are six different parameters involved here! To simplify things considerably, we consider the case where $p_r = p_c = p$, in which case it is reasonable to likewise assume $\tau_r = \tau_c = \tau$, with units such that these values are 1. After some algebraic manipulation, one can show that a sufficient condition for stability is that

$$0 \leq \mu_c < \frac{\mu_r - 2\tau/p}{2-p}.$$

The off-time of the cover session is something that we choose; these bounds give guidance how to choose this and avoid instability.

6. Conclusion

We are exploring a means of providing protection to traffic by obscuring its timing, and its data volumes. To accomplish this, one tunnels the traffic of interest inside of synthetically generated traffic. This approach will exact additional costs. It will delay the delivery of real traffic; indeed it is possible to increase its delivery time so much that the tunneling system cannot keep up with the demand for its services. It is important therefore to understand how to find cover traffic characteristics that are suitable, in both looking like real traffic, and in not creating so much additional overhead that the scheme is unworkable.

We approach the analysis problem using both simulation and an analytic model. We use the simulation to generate measurements of traffic behavior, using high fidelity models of protocols that carry traffic. Following development of the analytic model, we use those measurements validate it. The analytic model goes on to provide expressions for how much longer the tunneled traffic requires to be delivered, and for lower bounds on how idle the cover session can be and still keep up with real traffic demand.

Our paper shows that it is possible to take measurements from real or simulated traffic and use it to estimate parameters for an accurate analytic model of tunneling, *without the measurements or simulations having any notion at all of tunneling*. The nature of our problem allows for an interesting and useful decoupling of models. Analysis of the analytic model shows that its predictions of slowdown are very dependent on the accuracy of the ratios of of the mean time to transmit a packet using the cover session pattern, to the mean time to transmit it in the native real flow. The analytic model is useful as an explanatory device without accurate measurements of these means, but is questionable as a *predictive* device without them. Together these different model forms can address analysis of tunneled traffic in a way that neither can individually.

Acknowledgements

This work was supported in part by NSF Grants CNS-0524695 and CNS-0627671. Nabil Schear is also partly funded by Los Alamos National Laboratory.

References

[1] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11, 2005.

[2] G. Danezis. The Traffic Analysis of Continuous-Time Mixes. *Privacy Enhancing Technologies*, pages 35–50, 2004.

[3] D. M. Nicol, J. Liu, and M. Liljenstam. Simulation of Large-Scale Networks Using SSF. In *Proceedings of the 2003 Winter Simulation Conference*, pages 650–657, New Orleans, LA, December 2003.

[4] S. Ross. *Stochastic Processes,* 2^{nd} *Edition*. Wiley, New York, 1996.

[5] J.-Q. Shi, B.-X. Fang, B. Li, and F.-L. Wang. Using Support Vector Machine in Traffic Analysis for Website Recognition. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 5, pages 2680– 2684, August 2004.

[6] D. X. Song, D. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 25–25, Berkeley, CA, USA, 2001. USENIX Association.

[7] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[8] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. In *WOEC'96: Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, Berkeley, CA, USA, 1996. USENIX Association.

[9] C. Wright, F. Monrose, and G. M. Masson. HMM Profiles for Network Traffic Classification. *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and Data Mining for Computer Security*, pages 9–15, 2004.