Protecting Insecure Communications with Topology-aware Network Tunnels

Georgios Kontaxis Angelos D. Keromytis
Department of Computer Science
Columbia University
New York, NY, USA
{kontaxis, angelos}@cs.columbia.edu

ABSTRACT

Unencrypted and unauthenticated protocols present security and privacy risks to end-to-end communications. At the same time we observe that only 30% of popular web servers offer HTTPS. Even when services support it, implementation vulnerabilities threaten their security. In this paper we propose an architecture called Topology-aware Network Tunnels (TNT) which minimizes insecure network paths to Internet services without their participation. TNT is not a substitute for TLS. We determine that popular web destinations are collocated in a small set of networks with 10 autonomous systems hosting 66% of traffic. At the same time cloud providers own these networks or are very close to them. Therefore clients can strategically establish secure tunnels to these providers and route their traffic through them. As a result adversaries not able to compromise the web service or its hosting provider are presented with encrypted and authenticated traffic instead of today's plain text. The strategic placement of network tunnels, gathering of network intelligence and routing decisions of the TNT architecture are not found in VPN services, network proxies or Tor. Existing overlay routing systems such as RON and one-hop source routing cannot substitute TNT. We implement our proposal as a routing software suite and evaluate it extensively using diverse cloud and ISP networks. We eliminate plain-text traffic to the Internet for 20% of web servers, reduce it to 1 network hop for an additional 20% and minimize it for the rest. We preserve the original network latency and page load time. TNT is practical and can be deployed by clients today.

1. INTRODUCTION

Unencrypted and unauthenticated protocols present security and privacy risks to end-to-end communications. End-user Internet service providers (ISPs), including popular ones such as AT&T and Verizon, inject advertisements [22] and tracking headers [3] in the unencrypted HTTP traffic of their customers. They also tamper with SMTP traffic [4] to disable its opportunistic encryption. Intelligence agencies eavesdrop [8] on unencrypted traffic to piggyback on HTTP cookies for the purpose of tracking individual users. In addition, they impersonate popular Internet services through man-on-the-side attacks on unencrypted network paths for user exploitation and surveillance. Finally, network adversaries [21] inject JavaScript code to unencrypted HTTP traffic in transit to launch denial of service attacks.

Secure protocols, such as HTTPS, protect from these attacks however they are not widely deployed. Recent efforts to make HTTPS more affordable [2] and easy to deploy [1] are steps towards the right direction. Unfortunately a significant portion of web traffic still traverses the network without any security. In this paper we evaluate the security of 10,000 popular web sites and find that only 30% support HTTPS. We examine HTTPS-capable sites and discover that 53% let their visitors default to plain-text HTTP. Overall only 56 of the 10,000 sites fully protect their users through a combination of HTTPS and HSTS preloading. Even when HTTPS is available implementation vulnerabilities threaten its security. Recent attacks such as FREAK [14] and Heartbleed [11] have impacted hundreds of thousands of Internet services. End users depend on administrators of individual services to deploy HTTPS and keep their systems current. Unfortunately services may remain vulnerable for months after the disclosure of an attack [10].

In this paper we propose an architecture called Topology-aware Network Tunnels (TNT) which minimizes insecure network paths to Internet services without their participation. We argue that if end-to-end security with a server is not available the next best thing is a secure link to a network that is close to the server and will act as a gateway. We determine that popular web destinations are collocated in a small set of networks with 10 autonomous systems hosting 66% of traffic. At the same time cloud providers own these networks or are in close proximity. Therefore clients can strategically establish secure tunnels to these providers and route their traffic through them. As a result adversaries, such as ISPs and Internet backbone operators, not able to compromise the web service or its hosting provider are presented with encrypted and authenticated traffic instead of today's plain text. Making routing decisions given multiple traffic avenues is not trivial. We present the concept of topology-aware tunneling where a network overlay maintains secure links to multiple vantage points, considers the destination of traffic and routes packets through the tunnel maximizing content security. The strategic placement

CCS'16, October 24 - 28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978305

Architecture	Content Security	Anonymity
TNT	Complete	No
Tor	Partial	Yes
VPN	Partial	No

Table 1: TNT encrypts the entire path between the client and the destination network by optimizing tunnel placement and traffic routing. The same logic in Tor would compromise its anonymity properties.

of network tunnels, gathering of network intelligence and routing decisions of the TNT architecture are not found in Virtual Private Network (VPN) services, network proxies or Tor. Table 1 draws the distinction with these services and in section 3 we expand on their differences. Existing overlay routing systems such as RON and one-hop source routing cannot substitute TNT as they lack the tunnel-placement decisions and security-oriented metrics which are contributions of our work. Content delivery networks optimize their proximity to their clients but do not offer any security nor do they consider the path to the original server.

TNT enables users to take action when end-to-end security is not available and make sure their traffic is encrypted and authenticated as it transits the Internet. Our architecture is enabled by the rise in cloud computing and specifically infrastructure-as-a-service (IaaS) providers that are also used by most web services. The services themselves do not need to make any changes on their end and are not impacted in any way. While TNT is motivated by the prevalence of insecure communication protocols it can also benefit vulnerable implementations of secure protocols. HTTPS services vulnerable to attacks such as FREAK, using weak ciphersuites or expired certificates can be accessed through TNT to minimize exposure to a potential adversary. The decision to visit an HTTP server as well as a vulnerable HTTPS server over TNT is made transparently by TNT without the user's involvement. We implement TNT using TLS however in our architecture the client controls both endpoints of the connection. In HTTPS and similar protocols the web server controls just one of the endpoints. As a result we can update both ends should new TLS vulnerabilities arise. In HTTPS clients and servers patch their end independently. Moreover updating a TNT deployment benefits all the web services in the networks it covers. For the same effect, all services, e.g., in a cloud, must patch their individual systems. TNT is not a substitute for TLS. Unfortunately with the majority of web services incapable of TLS as well as a plethora of vulnerable and misconfigured TLS deployments TNT can benefit network security.

We implement our proposal as an IP routing software suite with negligible processing and memory overhead on the system. TNT can handle all transport and application layer protocols and thus protect traffic beyond HTTP, for instance SMTP. Realizing TNT is not a trivial process. It involves network-measurement as well as system-implementation challenges. The mapping of Internet routing must cope with some routers being uncooperative or adversarial towards active measurements by standard tools. At the same time implementing TNT on the client must offer the protection benefits of an inline system without connectivity disruption in case of failure. The nature of TNT entails updating the operating system's routing table in real time which if not done properly will break established TCP connections.

We evaluate our implementation of TNT extensively using a diverse set of cloud and Internet service provider networks. We find significant security benefits while preserving network performance. TNT provides confidentiality and integrity for the entire path between the client and the local network of the server for 20% of popular web servers. User traffic towards these servers is currently traversing the Internet in plain text. For an additional 20% of servers we are able to reduce network paths to a single hop. In general TNT reduces the number of distinct networks unencrypted traffic must traverse on the Internet by at least 33% in 70% of the cases and at least 50% in 40% of the cases. Overall TNT offers consistently shorter paths than ISPs. Shorter paths minimize the exposure of plain-text traffic to network adversaries. In terms of network metrics we do not deviate in latency from a fast academic network and are consistent in page load time for popular websites. Our implementation is practical and can be deployed by clients today by taking advantage of virtual machine platforms on cloud providers. Cloud providers could also offer TNT as a service similar to initiates by CDN providers enabling TLS for services they host [2]. TNT bypasses native traffic routing on the Internet and this may upset the load balance on the existing network infrastructure. However network provisioning is based on demand and our proposal, by design, funnels traffic to popular destinations. Accommodating TNT is aligned with improving connectivity to the cloud in general.

Our work makes the following main contributions;

- We identify the clustering of Internet services inside cloud providers and propose strategically establishing encrypted tunnels to their networks to avoid exposing plain-text traffic to the Internet.
- We define the following security-oriented metrics for routing traffic through our architecture; (a) Number of autonomous systems (AS) plain-text traffic must traverse to reach an Internet service. Ideally zero because the service is in the same network as a TNT exit. (b) Involvement of a particular trusted or untrusted AS.
- We implement and evaluate TNT as a IP routing software suite. We address non-trivial challenges in network measurements and system integration.

2. MOTIVATION

This paper is motivated by the limited presence of encrypted and authenticated communication protocols on the Internet. We focus on HTTPS and 10,000 popular web services according to Alexa. We find that only 30% offer HTTP over TLS. In practice 15% redirect to HTTPS. Just 4% of the sites have an HSTS policy that prevents an active network attacker from downgrading clients to plain HTTP. At the same time we observe that popular web services are collocated in a small set of networks with 10 autonomous systems hosting 66% of the traffic generated by a web browser when visiting the home page of 10,000 popular web servers. We argue that if clients can reach these few networks securely they are able to connect to the hosted web services without exposing their plain-text HTTP traffic to the Internet.

	HTTPS response	HTTPS?	%	#
1	Error (Conn. refused)	No	21.4	2144
2	Error (Invalid cert.)	No	22.1	2205
3	Error (HTTP 4xx 5xx)	No	2.9	292
4	HTTPS downgraded	No	21.5	2152
	Total	No	67.9	6793
5	OK	Yes	17.0	1695
6	OK (HTTP upgraded)	Yes	15.1	1512
	Total	Yes	32.1	3207

Table 2: HTTPS capability of 10K popular domains. Only 32.1% offer transport layer security.

2.1 HTTPS Adoption

To quantify the extent to which HTTPS has been adopted by Internet sites we evaluated 10,000 popular web domains according to Alexa.We focused on the `.com`, `.org` and `.net` top-level domains that resolved to US ASes. We verified the TLS certificates presented by these domains using the certificate authorities trusted by Mozilla. Table 2 presents our findings. Our HTTPS connection attempts were refused by 21.4% of the servers. Even worse, 21.5% redirected our HTTPS requests to HTTP. Additionally, 22.1% of the servers returned a TLS certificate which failed verification. Overall we failed to contact almost 70% over HTTPS.

The few sites supporting both HTTP and HTTPS need to make sure their visitors reach their secure endpoint. Search engine results and links from other sites might steer users towards the insecure HTTP. Also, if users omit the `https://` scheme when typing in the address bar, their browser will default to the insecure `http://`. Unfortunately only 47% of the HTTPS-capable sites (15.1% overall) redirect their visitors to HTTPS. For the majority of HTTPS-capable sites users will continue to visit them over HTTP. To make matters worse an active network attacker can prevent the redirection to HTTPS from taking place by replacing `https://` URLs with `http://` in the server's responses in flight. Some ISPs are known to remove the STARTTLS string from SMTP responses serving a similar purpose for e-mail. The use of the `Strict-Transport-Security` HTTP header can mitigate this by instructing the user agent to place future requests exclusively over HTTPS. We evaluated the use of HSTS among servers redirecting visitors to HTTPS and found that only 25% return a valid policy. Overall out of 10,000 popular web servers we find that only 3,207 (32.1%) support HTTPS and just 420 (4.2%) support HTTPS with an HSTS policy. Note that just 56 domains are found in the hard-coded HSTS preload list of Chrome and Firefox.

2.2 Web service collocation

To study the geography of Internet services we mapped the web sites from our data set to their respective ASes. A site may depend on more than one domains for resources such as scripts and images so we used a web browser to fully render the home page of each domain in our data set and recorded the destinations involved. We did not log HTTPS requests. We consider the home page of a domain to be the content received when visiting the exact domain or the standard `www` subdomain.

We visited the home pages of 9,944 domains from out data set. We excluded the 56 HTTPS-capable domains found in the HSTS preload list of Chrome. Ultimately we made 701,929 HTTP requests towards 34,893 unique domains to fully render the home pages. We subsequently resolved the domain names to their respective IP addresses and mapped them to ASes based on BGP prefix announcements collected by APNIC. Table 3 presents the top 10 most frequent ASes hosting the web servers involved in our 701,929 HTTP requests. The top 10 most frequent ASes host servers that receive 65.8% of all HTTP requests made.

%	Autonomous System Name
17.1	Akamai Technologies, Inc.
13.9	Amazon.com, Inc.
11.4	CloudFlare, Inc.
9.9	Google Inc.
3.7	EdgeCast Networks, Inc.
2.9	SoftLayer Technologies Inc.
2.1	Fastly
1.7	Tinet SpA
1.6	Internap Network Services Corp.
1.5	Rackspace Hosting
65.8	Total

Table 3: Top 10 most frequent ASes hosting sites.

Web servers hosted by Google present an interesting case. 22% of the HTTP requests made to Google servers target the `google-analytics.com` and 13% the `doubleclick.net` domain. As evidence has shown [8] passive network adversaries colluding with Internet backbone providers collect identifiers involved in requests to these domains to track users. It is also interesting that to reach Google our requests had to travel through two different tier 1 Internet backbone providers. The requests were made from a residential ISP and a university network in the US. In contrast, using our proposal (TNT) we can reach Google in a single network hop without exposing traffic to backbone providers.

To summarize, web services are clustered in few networks owned by cloud and other infrastructure-as-a-service (IaaS) providers. If end-to-end security with these services is not available, the next best thing is for users to establish a secure link to these networks and route traffic through it. Cloud providers make this approach practical as users can deploy their own virtual machines in the same networks.

3. RELATED WORK

To limit the exposure of their plain-text traffic some users connect to Virtual Private Network (VPN) servers offering encrypted tunnels between the client's device and some fixed point in the Internet beyond which traffic is unencrypted. While such services protect from a local network attacker, exposure to network adversaries might even increase as opposed to a direct route without the VPN service. Since VPN gateways are not optimized to be close to web servers user traffic might traverse more autonomous systems or even cross national borders, e.g., from a US gateway to a European server. Even VPNs with diverse gateways employ them without considering the destination of traffic.

Tor [15] is an anonymity network where traffic is encapsulated in layers of encryption and usually travels between three nodes before the original TCP/UDP packet exits to the Internet. By design Tor attempts no correlation between the exit of an encrypted circuit and the destination of traffic. Sherr et al. [24] propose the introduction of performance metrics in anonymous routing systems to affect circuit selection. Even though their work focuses within the anonymity network one could propose extending it so that exit nodes consider the destination of traffic. That would compromise the anonymity Tor offers. Even if someone were to give up anonymity we argue that the Tor network cannot carry user traffic close to Internet services. Using the Tor network status protocol we studied 1010 exit nodes with the highest bandwidth consensus weight and found that only 1% of them was located within major cloud networks. Meek [9] and domain fronting in general set up HTTPS tunnels to the cloud and CDNs and use them as gateways to masquerade the client's connection to a blocked website or Tor bridge. The placement of gateways ignores the location of the website and prefers networks an adversary is unlikely to unblock. As a result, Tor serves a different purpose than TNT and the two architectures complement each other. Table 1 summarizes their differences. Compared to the work of Sherr et al. we offer a complete implementation, evaluate it end-to-end, and introduce security metrics for routing decisions.

Overlay networks have been used in the past to recover from link failures in the underlying infrastructure and achieve better end-to-end performance. Savage et al. [23] propose Detour, an overlay where its members periodically exchange performance metrics such as RTT and packet loss and base their routing decisions on them thereby bypassing the Internet's native algorithms. LASTor [12] is a similar idea for Tor. Andersen et al. [13] in RON use a similar architecture to quickly route around link failures. Both designs limit their scope to members of the overlay and cannot be used for availability or performance guarantees for the rest of the Internet. Participating nodes evaluate each other on a regular basis which is something that does not scale well to the number of Internet services. Gummadi et. al [18] use one-hop source routing (SOSR) to recover from link failures in well-connected parts of the Internet. They maintain an overlay of virtual routers and clients use them as proxies to probe and connect to arbitrary network destinations when their default route is unable to deliver traffic. Our work differs from SOSR in two fundamental ways. In SOSR the placement of routers is selected at random since this is more likely to offer alternative links to most network destinations. In contrast, we optimize the placement of TNT routers in the edges of the Internet, inside cloud networks where Internet services form clusters. We also make routing decisions so as to carry encrypted traffic as close to the destination as possible while SOSR prioritizes finding any available path around a failed link. In terms of implementation TNT solves engineering challenges such as routing updates in the presence of active client flows while SOSR reroutes traffic that is already failing. In addition, our TNT router operates on IP packets while offering an interface for applications to select which traffic should be handled based on their own context. This is similar to RON which is otherwise incompatible with our design. LASTor [12] is a modified Tor client which uses a static AS-level map of the Internet to predict network paths and avoid circuits with the same AS at its edges. Astoria [20] follows a similar approach. In TNT we carry out data plane measurements to reliably construct the network path to a destination and reevaluate the path over time to account for routing changes. Moreover [12,20] focus on diversifying the ASes involved in the edges of a Tor path. TNT minimizes the length of the Internet path from a client to a server which is a different objective, optionally avoiding specific ASes.

4. THREAT MODEL

In our threat model the adversary can both passively monitor and actively alter network traffic at some point between a client and a server. This includes end-user ISPs as well as Internet backbone operators. Backbone networks, especially Tier 1 providers, are able to eavesdrop and tamper with traffic from multiple ISPs as it passes through. In section 1 we describe incidents involving both types of adversaries.

Clients on a public network, e.g., WiFi hotspot, run TNT locally on their system. Alternatively, in a trusted private network such as a residential setup TNT can run on the home router. We also consider the networks hosting Internet services as trusted. Hosting providers have a clear incentive to keep their network secure from external threats and honor their agreement with customers. Cloud networks where customer traffic may travel between data centers are assumed to secure their links. As a matter of fact Google has responded to evidence that intelligence agencies were eavesdropping [7] on its data centers by encrypting [5] the connections between them. Microsoft [6] has done the same. An adversary able to gain access to the trusted networks or systems of the client or the server is out of scope.

TNT creates encrypted tunnels between a client and key networks where Internet services form clusters, namely in the cloud. As a result adversaries not able to compromise the services or their hosting providers are presented with encrypted and authenticated traffic as opposed to plain text. This includes end-user ISPs and Internet backbone operators which are presently a threat because of their position in the network. It might seem that TNT exits create appealing targets where Internet traffic is funneled through a few networks. However, an adversary able to monitor tunnel exits is already able to monitor the networks hosting the servers and gains no advantage by the presence of TNT.

5. ARCHITECTURAL OVERVIEW

We present an architecture called Topology-aware Network Tunnels (TNT) which minimizes insecure network paths to Internet services without their participation. An insecure path is a set of links over the Internet carrying traffic of unencrypted and unauthenticated protocols such as HTTP or SMTP. Shorter insecure paths limit the exposure of plaintext traffic to passive and active network adversaries. At a high level, TNT establishes a network overlay of secure tunnels between the client and a set of vantage points. TNT evaluates the network path from each vantage point towards each packet's destination. It then selects the tunnel minimizing exposure to adversaries.

The TNT architecture addresses two key challenges: (1) optimize the placement of secure tunnels across the Internet and (2) determine the optimal tunnel to route each network packet through.

5.1 Topology-aware Network Tunnels

The key intuition behind our proposal is that Internet services are clustered in few cloud and infrastructure-as-a-service (IaaS) providers. Therefore we can optimize the number and placement of secure tunnels by collocating them

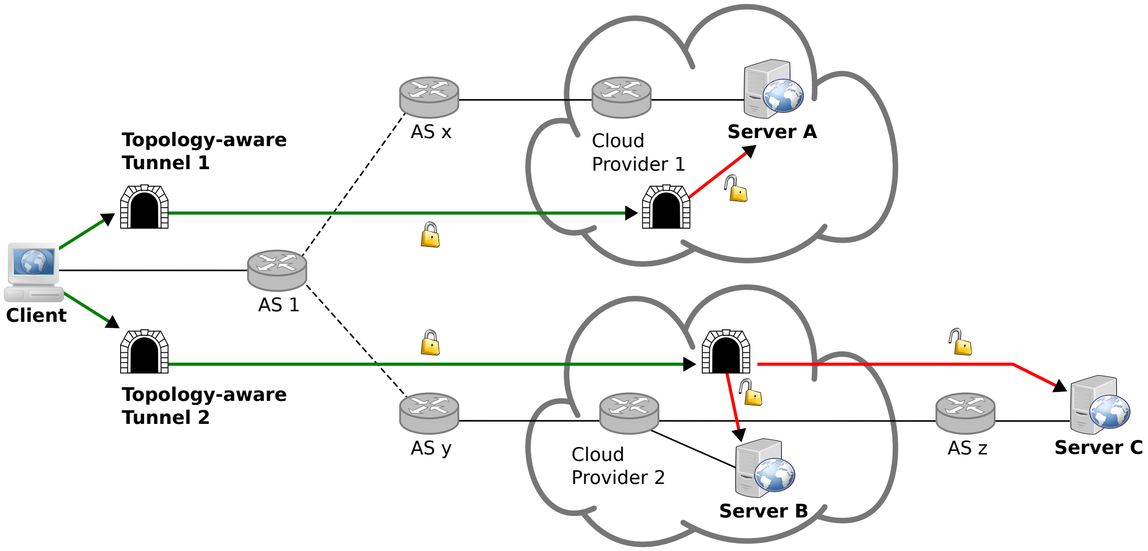

Figure 1: In the TNT architecture an overlay of secure topology-aware tunnels is established between the client and a set of network vantage points. The number and placement of secure tunnels is strategically selected to minimize the network distance packets need to travel outside the overlay to reach their destination. Individual network packets are intelligently routed through the tunnel exiting closest to their destination. Tunnel exits within the same network as the destination of a packet (Servers A, B) eliminate the exposure of traffic to network adversaries.

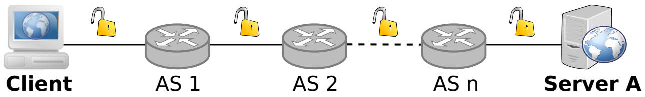

Figure 2: Example of a network path on the Internet. For insecure protocols such as HTTP data are exposed across the path to operators of the underlying infrastructure.

with these infrastructure providers. This addresses the first challenge from above. That way we can shorten the insecure network path and essentially bring the client as close to these servers as possible, ideally within the same network. As a result, the traffic of insecure protocols will have minimal or zero exposure on the Internet. Apart from minimizing the overall path length, we also define metrics rewarding or penalizing the presence of a trusted or untrusted intermediate network in the path. The trustworthiness of a network is context specific so in this paper we focus on path length.

Figure 2 presents an example of a network path today. The set of links and routers a client's packets must traverse to reach a server is grouped into autonomous systems (ASes) and controlled by distinct organizations. Note that such path might span different countries or continents. This translates to potential passive and active attacks against the user's web browsing or e-mail.

Figure 1 presents the TNT architecture as an overlay on the existing Internet infrastructure. TNT has established secure tunnels between the client and two cloud networks that exhibit high clustering of Internet services. `Server A` is hosted by `Cloud Provider 1` and we can reach it through `Topology-aware Tunnel 1` without exposing plain-text traffic to the Internet. Packets towards `Server A` enter the tunnel before leaving the user's network and are encrypted and signed. Internet routers operated by `AS 1` and `AS x` observe an encrypted flow from the user to `Server A`. Without TNT these ASes have access to plain-text traffic. Packets exit the tunnel inside the trusted network of `Cloud Provider 1` and are authenticated and decrypted. Subsequently packets transit the cloud provider's network and reach `Server A` which is unaware of the process. To reach `Server C` without TNT the user's packets will travel in plain text through `AS 1`, `AS y` and `AS z`. With a TNT link to `Cloud Provider 2` they travel encrypted and signed through `AS 1` and `AS y`. `Server C` is an outlier not hosted in a cluster of Internet services. In this case TNT is able to minimize the length of the insecure network path so instead of 3 ASes only `AS z` will be able to observe plain-text traffic. Next we describe how the TNT router determines the optimal tunnel to route traffic through so as to minimize insecure network paths.

5.2 The TNT Router

The TNT router is a routing software suite managing topology-aware tunnels and directing traffic through them. It is located on the client's system or local network gateway, for instance a home router, and maintains topology-aware tunnels with remote networks based on the placement strategy described earlier. It has a network-mapping and a decision-making component. Given the available tunnels and a specific destination address the mapping component employs a set of probes to discover the network path between each tunnel's exit on the remote network and the destination. The discovery process involves active and passive network measurements described in section 7. The information is passed on to the decision-making component which evaluates it and assigns metrics on each tunnel based on its suitability to carry traffic to the specific destination. Based on the metric the TNT router directs outgoing traffic through the tunnel which minimizes its value. This satis-

```
Kernel IP routing table                                   (1)
Destination  Gateway      Genmask          M IF
0.0.0.0      192.168.0.1  0.0.0.0          0 eth0
192.168.0.0  0.0.0.0      255.255.255.0    0 eth0
```

```
Kernel IP routing table                                   (2)
Destination  Gateway      Genmask          M IF
0.0.0.0      10.0.0.1     0.0.0.0          0 tun0

a.m.z.n      192.168.0.1  255.255.255.255  0 eth0
a.z.u.r      192.168.0.1  255.255.255.255  0 eth0
192.168.0.0  0.0.0.0      255.255.255.0    0 eth0

10.0.0.1     0.0.0.0      255.255.255.255  0 tun0
10.0.1.1     0.0.0.0      255.255.255.255  0 tun1

a.b.c.d      10.0.1.1     255.255.255.255  1 tun1     (5)
```

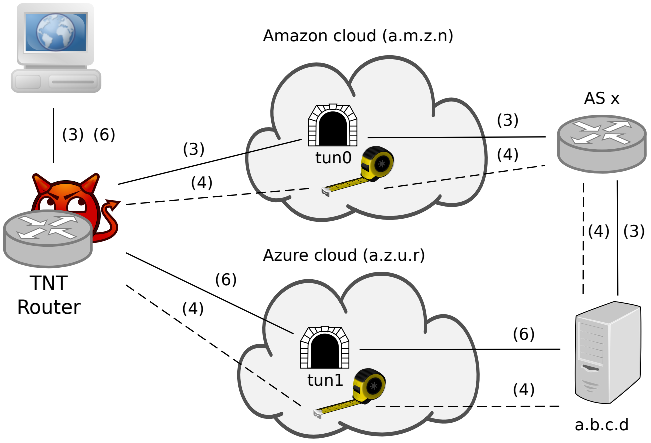

Figure 3: Operation of the TNT router when serving client request towards network destination a.b.c.d. Initially it updates the system's routing table (1) to route all network packets through one of the tunnels by default (2). A client's request for which there is no explicit route will go through the default tunnel (3). Subsequently the TNT router will task the probes at each tunnel's exit with determining their distance from that destination so that future requests can be better routed (4). Following the path announcements from the probes an explicit routing entry is created for that destination (5). The operating system will use that entry for future client requests (6).

fies the second challenge from the beginning of this section. To account for the dynamicity of Internet routing the TNT router periodically reevaluates these metrics.

6. IMPLEMENTATION

We have implemented the TNT router as an IPv4 routing software suite and tested it in Linux. An Internet-layer implementation is more flexible since it is transport and application-layer agnostic. While the core of the router operates at the IP layer, peripheral components implement high-level logic that enables traffic handling based on transport and application-layer heuristics. By default the router focuses on HTTP (TCP port 80) and SMTP (TCP port 25) traffic while all other traffic, including HTTPS, is routed as if TNT is not in place. The TNT router presents its tunnels as network interfaces to the operating system. The router determines the optimal tunnel to route outgoing traffic through and communicates its decision to the operating system. It does so by interacting with the underlying routing structures. Updating the operating systems routing structures affects how IP packets are transmitted through the available network interfaces. The operating system ultimately writes outgoing packets to the appropriate interface.

The TNT router reacts to outgoing traffic but instead of preventing its transmission until it makes a routing decision it applies its decision to future flows to the same destination. This way it does not disrupt the user's activity or impact network performance. While this means that the first flow to a new destination is not routed optimally in section 7 we show that the routing state quickly becomes optimal following the router's initialization phase. The routing state persists across system restarts.

Realizing the TNT router addresses the following challenges: (1) Be practical to deploy, use and maintain. (2) Reliably discover the network topology between tunnel exits and Internet destinations to make routing decisions. (3) Dynamically update the system's routing table without disrupting existing connections. Any naive routing update will reset connection-oriented protocols such as TCP.

6.1 Deployment

We use OpenVPN to establish TLS-based tunnels with virtual machines in the cloud. Tunnels appear as network interfaces to the operating system with a standard 1500-byte MTU. IP packets entering the tunnel are handled by OpenVPN which fragments them if necessary, encrypts them, appends its signature and sends them to the other end of the tunnel which reverses the steps. OpenVPN supports a variety of ciphersuites from OpenSSL. Our design is not specific to a tunneling technology or ciphersuites.

The deployment of the TNT overlay is automated including installing the TNT router locally and launching the necessary virtual machines in the cloud. We use a combination of Unix shell scripts and the command-line interfaces offered by cloud providers. At the moment we prompt the user for their cloud account credentials however we envisage a deployment process without any user involvement. We do not depend on specific cloud providers but deploying a virtual machine is a provider-specific process which we need to implement. We launch Linux virtual machines in the cloud and configure OpenVPN on both ends of each tunnel. Once the tunnels are established the deployment phase is complete and the TNT router begins running on the user's system without the need for further interaction. Signing up for a new cloud account can be streamlined as part of the deployment script. Users do not share their virtual machines with others but they do share the underlying physical hardware. Attacks from a collocated virtual instance are beyond the scope of our threat model. Cloud operators could offer TNT links to their network as a service so that users do not need their own virtual machines.

6.2 Operation

The TNT router has three components; (a) a TNT traffic selection program running on the client, (b) the TNT routing daemon also running on the client and (c) probes running on the remote end of each tunnel, which in our case are virtual machines in the cloud. Figure 3 depicts the operation of TNT. Initially the TNT daemon brings up the tunnels as distinct network interfaces. One of them at random is set to be the default interface meaning that all traffic TNT is configured to handle goes through it. The system's routing table is updated from step `1` to `2` in the figure. Traffic TNT is not configured to handle gets routed as if TNT is not in place, i.e., still gets routed based on step `1` in the figure. So far all tunnels but the default remain unused by the operating system since it has no reason to prefer them over the default. As a result the client's initial requests to the Internet host `a.b.c.d` will go over the default tunnel (step `3`). Delaying outgoing traffic until the TNT router calculates the metrics for a destination would impact performance. Setting up a default TNT route instead protects traffic from end-user ISPs without the need to wait for a routing decision. In section 7 we show that the router quickly makes an optimal decision applied to subsequent flows.

The TNT traffic selection program inspects outgoing traffic for the purpose of identifying destinations that the TNT router must handle. It uses libpcap and is able to identify traffic flows. To select traffic it uses BPF expressions and by default focuses on outgoing TCP flows to port 80 so as to select HTTP traffic. For each new flow matching the selection filter it extracts the destination IP address and queries the TNT routing daemon for an optimal route. If not found it tasks the forward probes with with mapping their network path to the destination (step `4`). The forward probes subsequently communicate their findings to the routing daemon directly. To facilitate the network measurements the traffic selection program supplies not only the target IP address but additional context such as the transport protocol and destination port used. The forward probes listen on their end of the tunnel interface for control commands. They use active network measurements to discover the path to a destination on demand and announce it back to the router. We describe the measurement methodology in section 7.

The routing daemon interacts with the other components in two ways; it looks up IP addresses in the current routing table on behalf of the traffic selection program and evaluates network path vectors received from the forward probes. The traffic selection program queries the daemon with the IP addresses of destination in outgoing traffic flows. An IP address found in the routing table associated with a metric value means the optimal tunnel to reach that particular destination has been determined in the past. Otherwise the traffic selection program receives a negative response. The forward probes, tasked with discovering their network path to a destination, announce it back to the daemon. When evaluating the network path from a particular tunnel exit the daemon calculates a metric value based the tunnel's suitability to carry traffic to that destination. In this paper we focus on path length as our metric so the router the metric value is the number of ASes on the path. It subsequently decides to route traffic through the tunnel which minimizes the metric and updates the operating system's routing structures to direct packets for that destination through a particular tunnel interface rather than the default. In Linux the daemon uses the Netlink[1] interface to access and alter the necessary operating system structures. In figure 3 the probes announce paths with distance 2 and 1 respectively for `a.b.c.d` so a decision is made to route the destination through `tun1`. Future requests for that host will go through `tun1`. This is done with an explicit entry in the operating system's routing table (step `5`). Note that the newly introduced route will only be applied to flows matching the context this route was generated. So a route generated because of an outgoing TCP port 80 flow will only be applied to flows to that port. Flows to 443 or some other port to the same destination will not be affected. Applying a route only to specific transport or application-layer flows is discussed later in this section.

The TNT router performs a series of optimizations to its routing table. To avoid stale routing entries it implements a decaying system where entries that have not been used for routing recently are pruned from the routing table. Similarly, for frequently used destinations it schedules lazy reassessments of the optimal path with exponential backoff to stay current with Internet routing changes. TNT routing entries actually describe entire AS prefixes rather than individual hosts. When the forward probes respond to a mapping query for a particular destination host they lookup and return the related BPG prefix the destination's AS is responsible for. This eliminates additional queries for addresses in the same prefix. Finally the router aggregates routes by grouping adjacent route prefixes to form shorter prefixes and reduce the number of entries in the table.

6.3 Transparent Routing Updates

Updating the routing table of a live end-user system to essentially implement multihoming is not a trivial task. Network routers dynamically change their routing table on a frequent basis without the same challenge because they simply forward IP packets without altering their header. However, packets exiting an interface in an end-user system adopt[2] that interface's IP address as their source. A routing update directing packets of an existing TCP connection through a different interface will change their source IP address. Packets with the new source IP address will be dropped or met with packets with the RST flag set since from the remote endpoint's perspective do not match any existing connection. Ultimately TCP connection will close unexpectedly.

To ensure non-disruptive updates to the operating system's routing table we implement a transitioning process which guarantees the continuity of existing sessions in TCP as well as UDP and ICMP logical sessions. We utilize the support for multiple routing tables in the Linux kernel as well as the functionality offered by its Netfilter framework. The key idea is to split a routing update into two phases. Initially a new route is taken into consideration only for new connections while existing ones are routed as if the update never took place. This guarantees continuity. Eventually connections predating the update will terminate naturally and the system will reach a stable state where all connections use the updated route and the old one is deleted.

To implement this two-phase routing we clone the currently effective routing table into a new routing table and instruct the operating system to look up new connections in the new table while keep reusing the old one for existing con-

[1] http://lxr.linux.no/linux+v3.19/net/core/rtnetlink.c

[2] While there are ways around that, upstream providers usually implement egress filtering to block outgoing packets with spoofed IP addresses. In TNT different tunnel interfaces are expected to exit in disjoint network prefixes.

			Effective Routing Table by Connection Status[1]	
t	TNT Event	Routing State	Existing[2]	New
0		Stable	Main	Main
1	Start	Converging	Main	*TNT0*
2		Stable	*TNT0*	TNT0
3	Update route X	Converging	TNT0	*TNT1*
4		Stable	*TNT1*	TNT1
5	Terminate	Converging	TNT1	*Main*
6		Stable	*Main*	Main

[1] *Connection status is independent of the transport protocol used. It is logical, applies to TCP as well as UDP and ICMP and is based on timers and bi-directional IP packet exchange.*

[2] *Existing connections are in a logically-assured state. For TCP this is either the Related or Established state.*

Table 4: Updating the operating system's routing table so as not to disrupt existing TCP connections. The TNT router transitions the system from the current table to an updated version by cloning the table, modifying the new table and setting it as effective only for new connections.

nections. Initially we clone the default, main routing table into a new table TNT0. (Time `t0` in Table 4) The system transitions into a state where any already established TCP connections, as well as logically-assured UDP and ICMP connections, keep using the main routing table whereas the destinations of new connections are looked up in the TNT0 table. (`t1`) Eventually all connections predating the update (`t1`) will naturally terminate and the system will reach a state where all current and future connections will use table TNT0 exclusively. (`t2`) Subsequently any updates after time `t2` will clone TNT0 into TNT1, enter a converging state `t3` and eventually reach a stable state `t4`.

If we need to update the effective routing table while the system is still converging from a previous update we must allocate an additional table instead of recycling an existing one. For example a new update during time `t3` will cause the effective table TNT1 to be copied to a new table TNT2 which will then be updated and marked as the effective table. We cannot reuse table TNT0 at this time since it still being used by connections predating the last routing update. We try to carry out updates in batches to avoid the need for more than two tables at a time. However traffic scenarios such as web browsing might cause bursts of updates that do not fit in a single batch. Under reasonable conditions Linux does not limit[3] us in the number of tables we can maintain. As soon as all connections associated with an old routing table are terminated that table becomes eligible for reuse in a future routing update. In section 7 we quantify the amount of routing tables necessary under realistic network activity. A routing cache would eliminate the need for the above technique. Since version 3.6 [19] the Linux kernel no longer supports such cache for efficiency reasons. Windows 7 implements a routing cache but the same reasons might justify its removal from future versions.

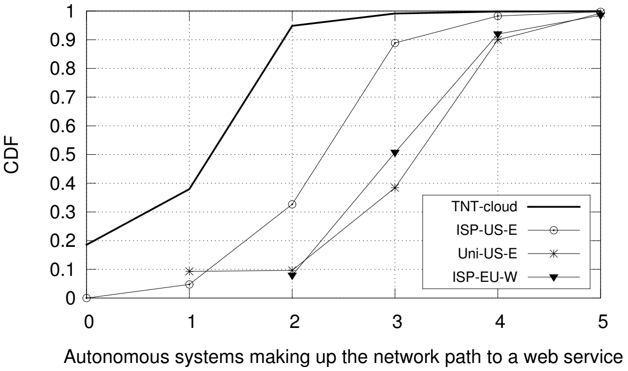

Figure 4: CDF of the number of ASes on the network path to each web service. TNT outperforms ISPs by exposing zero traffic to the Internet for 18.5% as well as achieving one-hop paths for an additional 19.5%.

To advise the operating system which routing table to use for each destination lookup we use kernel routing policies. Each routing table is associated with a mark and we use Netfilter's connection tracking to label new connections with the mark corresponding to the new table. Policies match the mark individual packets carry to specific routing tables.

6.4 Application-specific routing

As mentioned earlier both the operating system's core routing functions and the core of the TNT router make IP-based routing decisions. At the same time it makes sense to configure traffic routing preferences based on high-level context coming from the transport and application layer. For instance by default TNT must only handle IP packets belonging to TCP port 80 flows (HTTP) while HTTPS and any other traffic must not be affected. By default Linux uses a global routing table which affects all packets and is not suitable to our needs. To achieve the necessary flexibility we use routing policies which are combined with multiple routing tables and the Netfilter framework. Using the latter we mark specific connections or packets based on heuristics such as destination port. Marked packets subsequently are matched to specific routing policies leading to corresponding routing tables. For example we have a IP routing table that is only used for TCP flows to port 80. The system's default table is never modified and, unless we explicitly mark outgoing packets, traffic is routed as if TNT is not present.

7. EVALUATION

7.1 Network Proximity

We quantify the exposure of plain-text traffic to adversaries adversaries by mapping the network paths to popular web sites using a series of Internet vantage points. We then compare the results to a TNT deployment to the AWS and Azure cloud networks to evaluate the ability of TNT to minimize traffic exposure. For our measurements we used a total of 7 vantage points spread across the US and western Europe; 4 virtual machines in the Amazon Web Services

[3] Since version 2.6.19 the Linux kernel supports up to 2^{32} routing tables and efficiently addresses them using a hash table. Previous versions supported up to 255 routing tables.

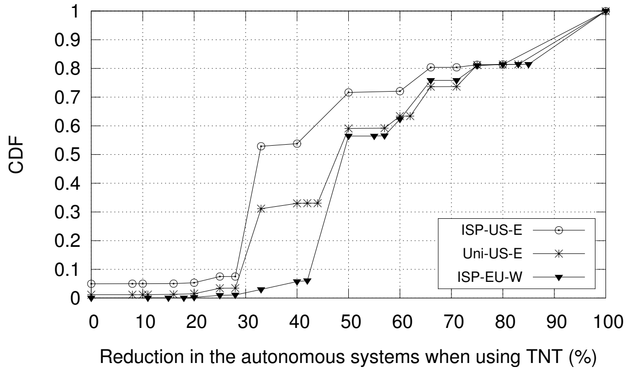

Figure 5: CDF of the reduction (%) in ASes on a network path when using TNT as opposed to an ISP. TNT offers at least 33% in 70% of the cases.

(AWS) and Microsoft Azure (Azure) cloud, 2 end-user lines in ISPs and access to a fast academic network. Our set of hosts was compiled by visiting the home page of 9,944 popular web domains according to Alexa with phantomJS, a Webkit-based, Javascript-capable headless web browser, and collecting HTTP requests. Our final list, including the initial domains, contains 34,893 unique domains resolved to 20,026 distinct IP addresses.

Our network mapping process correlates active network measurements with BGP routing views [16]. This is the same process followed by the forward probes of the TNT router to discover network paths. To measure the actual flow of packets between one of our vantage points and each web server in our data set we sent ICMP type 8 as well as TCP+SYN 80 packets to the destination host and elicited ICMP type 11 responses packets from all intermediate network devices using varying TTL values in the header. Some network policies drop ICMP packets and firewalls at the destination might drop all packets but the ones the service is expecting. By using port 80 for web services we guarantee minimal disturbance for our measurements. In section 6 we discussed how forward probes of the TNT routing receive the same information so as to adjust their measurements to packets that are guaranteed to reach their destination undisturbed. Nevertheless some network policies silently drop TTL-expired packets so we only considered complete network paths for which we had identified all their hops ending up with data for 15,020 of the original 20,026 hosts. We subsequently resolved the IP addresses of each hop in the network paths to their respective AS using BGP prefix announcements collected by APNIC. Finally we verified the accuracy of our measurements by correlating the derived AS paths with BGP views [4] from looking glass platforms. Our methodology guarantees an accurate picture in the case of transit, i.e., customer-provider, relationships between ASes while leaving a margin of error in the case of peering agreements where BGP announcements are not available outside the participating ASes. In production the TNT router will ignore AS paths produced by active measurements that cannot be verified through BGP announcements.

Figure 4 presents the CDF of proximity, in terms of ASes involved, of each vantage point to the web hosts in our data set. We define this as our exposure metric, indicating the number of potential network adversaries, which the TNT architecture aims to eliminate or minimize. A distance of zero ASes in the figure translates to the server being in the same AS as our TNT link. Similarly a distance of one AS indicates a direct, peering relationship between our trusted AS and the AS of the server. One may observe that the TNT architecture outperforms end-user ISP and university networks by routing packets to 18.5% of destinations through ideal, adversary-free, paths. Note that TNT also outperforms the individual cloud networks we used. Zero hop network paths in the case of ISPs are attributed to CDNs hosted in their networks and occurs in less than 0% and 5% of the cases respectively. Since end-user ISPs are part of our threat model we do not consider these paths adversary-free. We have also calculated the average proximity of TNT to the home page of each domain as a whole, including all subresources. The results are consistent with figure 4. Similar results describe the proximity of TNT to advertisement networks. Additionally, TNT offers consistently shorter paths to almost all destinations tested. Figure 5 shows that we achieve at least 33% and 50% shorter paths in 70% and 40% of the cases.

From a privacy perspective one might be skeptical about funneling their traffic to a few large cloud providers. Note that this traffic is already transiting the public Internet in plain text. As TNT scales traffic is distributed closer to its destination and user privacy improves. Nevertheless one could use TNT to only access destinations that are hosted in the clouds it maintains tunnels with, which is almost 20%.

7.2 Operating a TNT router

The TNT router is our implementation of topology-aware tunneling for clients to use. To quantify its impact on the end user's system or other network gateway we carried out a web browsing session that generated realistic network traffic patterns for the router to handle. We instrumented Firefox to visit in succession the home pages of 1,000 popular web domains according to Alexa. Firefox waited for each page to fully load before moving on to the next and we cleared its cache between sessions. HTTPS traffic was unaffected by TNT and was routed through the default interface. Plaintext HTTP was dominant as shown in section 2.

Our evaluation focuses on the impact the router has on the system's resources and is expressed in hits in the routing table, the number of concurrently active routing tables as described in section 6 and the number of entries found in the effective routing table over time. The first measure determines the amount of active network measurements necessary. The second and third measures determine the stress on the system's CPU and memory.

Initially we measured how quickly the operating system's routing table converged to its optimal configuration so that each IP packet is routed through the tunnel that exits closest to its destination. An IP packet with a destination address for which we do not yet have an explicit route is classified as a lookup miss and results in forward probes mapping and assessing the path to that destination. On the other hand, a destination address for which TNT knows the best way to reach it has an entry in the routing table and constitutes a lookup hit. Figure 6 presents the hit ratio over time. One may observe that approximately for the first 500 connections TNT has enough optimal routes to satisfy 50% of the destinations involved. This indicates a fast bootstrap

[4] `http://as-rank.caida.org`, `http://lg.he.net/`

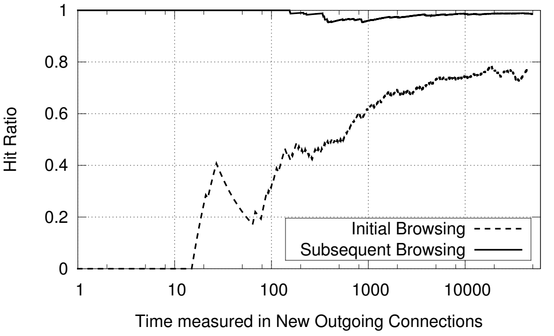

Figure 6: Ratio of optimal versus suboptimal TNT routing over time. Initially the ratio is low causing forward probes to map network paths. Later on it quickly rises indicating that few popular destinations have been mapped.

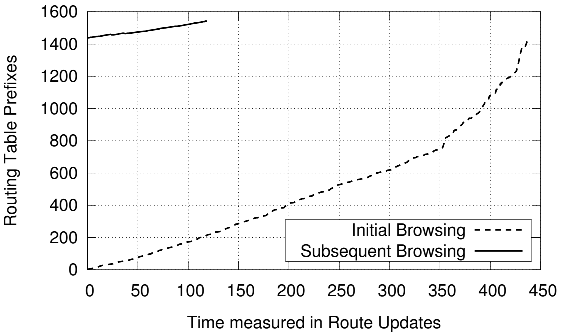

Figure 7: Number of entries in the system's effective routing table. The TNT router creates explicit entries per AS as part of its operation. In practice memory consumption is negligible and processing time near constant.

phase. Over time the hit ratio increases and by the end of the browsing session we see that TNT was able to satisfy almost 80% of the destination lookups. For subsequent browsing sessions the hit ratio remains between 100% and 98%. The slight drop is attributed to sites with dynamic content.

The TNT router adds a network-specific route, associated with a metric, for every destination the distance to which has been determined by the forward probes. As the user visits more and more unique Internet destinations the routing table grows. Web browsing is a representative example of this scenario as it involves a plethora of different servers. Figure 7 presents the number of routing entries in the effective routing table over time. Note that the effective routing table is the one the operating system will use to look up the destinations of new connections. During the initial browsing session the size of the routing tables reaches approximately 1400 entries. That might seem daunting compared to the initial 2 entries for most systems with a single network interface. However the implementation of the routing table (`fib_table`) in Linux is highly efficient[5]. It uses hash tables to lookup destinations in near constant time. The only way the number of entries impacts the system is in terms of memory consumption. However the way route information is stored in data structures is also efficient as it groups common parameters between routes to a single data structure (`fib_info`) that is shared by all routes. In practice the memory overhead for the number of routes TNT introduces is negligible even for embedded systems such as home routers. Note that we periodically expire routes that have not been recently involved in lookups.

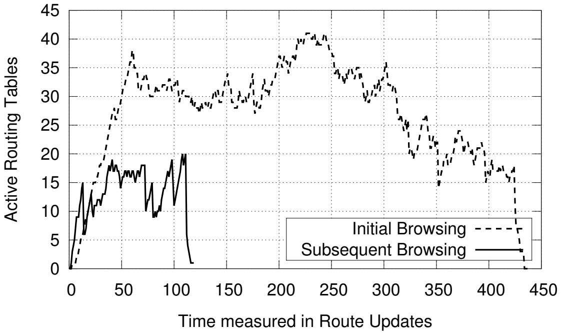

Figure 8: Number of routing tables concurrently active. Following an update existing connections keep using the previous version the table to avoid disruptions. In practice memory and processing overhead are negligible.

In order to ensure a smooth transition when updating the routing table the TNT router uses auxiliary tables as described in section 6. Visiting a page causes multiple connections to be created in an asynchronous bursty manner which may result in an equally bursty set of routing table updates. Figure 8 shows how the system converges from multiple routing tables to a single one. Multiple tables are used only during routing updates which are infrequent. We argue that web browsing models the worst case scenario in terms of traffic patterns and so this figure sets an empirical upper bound on multiple routing policies and tables.

7.3 Web Browsing over TNT

To quantify the effect TNT has on the web browsing experience we studied the round-trip time (RTT) of packets towards the respective servers along with the overall load time for each page. Note that we measured RTT from the client's perspective. Her packets had to traverse a TNT link, reach the cloud network and then proceed to their final destination. Our baseline was an academic network with a fast Internet connection in the east coast of the US. In terms of network latency, as figure 9 shows, TNT offers comparable times to our baseline. We sent TCP packets to destination port 80. In terms of page load time figure 10 shows consistent results between TNT and the baseline.

8. SECURITY DISCUSSION

For destinations hosted in the cloud a passive Internet adversary sees an end-to-end encrypted connection. Examples are servers A and B in figure 1. Such destinations are the primary use case for TNT so clients can reach them without exposing plain-text traffic to the Internet. Optionally, TNT may also leverage the cloud as a gateway to reach arbitrary Internet destinations over minimal unencrypted paths. Such

[5] http://lxr.linux.no/linux+v3.19/include/net/ip_fib.h

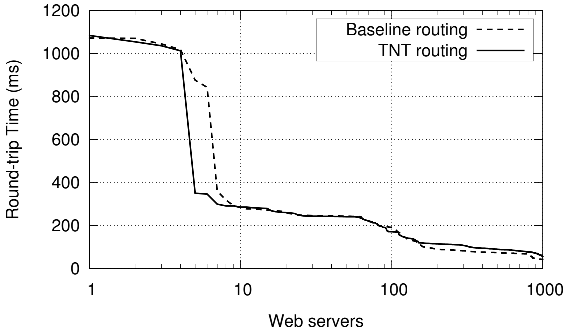

Figure 9: RTT for packets routed either through a fast academic network or a TNT link to AWS.

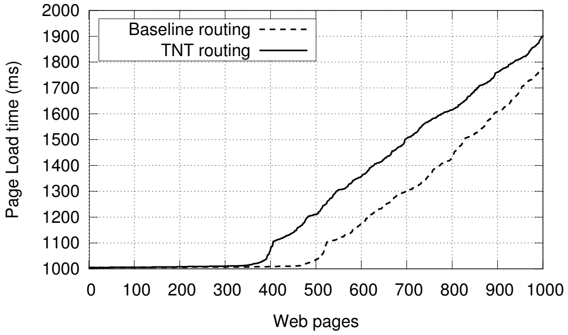

Figure 10: Load time for sessions routed through a fast academic network or a TNT link to AWS.

example is server C in figure 1 where only the path between the cloud and the server is unencrypted. We have shown that in such cases TNT always creates shorter paths. There is however a tradeoff between reducing the number of ASes observing plain-text traffic and routing it through networks that may not have originally observed it. Especially ASes adjacent to the cloud may seem at an advantageous position to monitor the browsing behavior of TNT users. However we do not observe any notable deviation in the shape of the frequency distribution of ASes involved when TNT is present. Without TNT the most frequent AS is found in 22.5% of the paths and with TNT the most frequent AS is found in 19% of the paths. These are two different ASes and naturally, because of our routing decisions, some ASes see more and others less traffic. However, as far as users are concerned there is no single AS that can observe more of their browsing history than without TNT. This can be explained by cloud providers having multiple upstream providers for redundancy and load balancing reasons. In fact, the diversity of ASes involved actually increases.

When the cloud is used as a gateway by TNT it appears to be the source of clients' traffic at the IP level. This facilitates TNT routing. We do not attempt to hide or anonymize the source or destination of traffic. ASes observing encrypted client traffic entering the cloud and unencrypted traffic exiting the cloud can attribute cloud-originating traffic back to a particular client. An adversary can use the timing and size of packets to match encrypted flows between clients and the cloud to plain-text traffic between the cloud and external sites. Cover traffic and shaping techniques may obfuscate such heuristics. However, we argue that the actual content of plain-text traffic carries a plethora of information that can identify users. For instance HTTP cookies, referrers and search terms are much more reliable in tracking users than the IP address of the device originating the traffic.

An active adversary could try to either block our ability to map network paths or falsify the data we receive. It could also try to block TNT links. To make TNT network measurements resistant to blocking we tailor our probes to the packets a specific service is expecting to receive. For HTTP we transmit IP packets with a TCP header indicating destination port 80. An adversary blocking such packets would also stop actual user traffic towards a service. Fingerprinting traffic generated by TNT measurements is possible though. Instead of blocking our network measurements an adversary could tamper with the data we receive by spoofing responses from upstream routers. However in section 7 we describe how we correlate network paths resulting from data plane measurements with AS paths from BGP announcements. A measured path that is infeasible is not taken into consideration by TNT. Attacks against BGP are beyond the scope of this work and any solution is orthogonal. An active adversary situated between the client and the cloud could try to prevent TNT links from being established. Our threat model does not include censorship and failure to run TNT in a network should warn users about the operators intentions.

Finally it might seem that TNT centralizes traffic flows within a few cloud networks which become appealing targets. However our threat model focuses on adversaries that are not powerful enough to attack the cloud but can carry out passive and active attacks today because of their location on Internet. This includes ISPs and other infrastructure operators. Moreover the key idea behind TNT is to utilize cloud networks to reach destinations already hosted within them. Therefore adversaries powerful enough to attack the cloud gain no advantage from the presence of TNT. As an additional, entirely optional, use for TNT we propose routing traffic to Internet destinations outside the cloud as to minimize the network path to them. While this makes such traffic available to adversaries capable of compromising cloud networks we argue that the benefit of shielding plain text traffic from every other adversary on the Internet presents an appealing tradeoff. At the same time the TNT architecture benefits from and encourages scaling to more cloud networks. We thus expect that individual clouds will see a decrease in the traffic going through.

9. LIMITATIONS

TNT maps network paths using IP-based measurements. It cannot identify hops operating below OSI layer 3 such as in the case of MPLS tunnels. As a result it will misrepresent the length of network paths featuring such traffic encapsulation. This is not a limitation of the TNT architecture but a constraint imposed by our implementation of network measurements. Gueye et al. [17] can approximate the geographical location of an IP host in the presence of loaded links. They use a set of known landmarks to compare the perceived end-to-end network delay to expected propagation time of the underlying physical links. Using the existing TNT distributed architecture we could approximate the location of network hops on the Internet and highlight hops that appear adjacent at layer 3 but are separated by a great physical distance such as in the case of an overseas or cross-country link.

10. CONCLUSION

In this paper we enable users to minimize the exposure of their plain-text traffic to Internet services without their participation. We find that most web servers are clustered in the networks of cloud providers. Therefore we propose to strategically establish secure tunnels to these networks and intelligently route traffic to them. We have implemented TNT as a routing software suite and can eliminate plain-text traffic over the Internet for 20% of web servers, reduce it to 1 network hop for an additional 20% and minimize it for the rest. The proliferation of cloud providers makes it practical for users to deploy TNT.

11. ACKNOWLEDGMENTS

We thank Roxana Geambasu, Vasileios P. Kemerlis and Michalis Polychronakis for early discussions and feedback. This work was supported in part by the National Science Foundation through Grant CNS-13-18415. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or the NSF.

12. REFERENCES

[1] A Certificate Authority to Encrypt the Entire Web. https://www.eff.org/deeplinks/2014/11/certificate-authority-encrypt-entire-web.

[2] Cloudflare - Introducing Universal SSL. https://blog.cloudflare.com/introducing-universal-ssl/.

[3] Electronic Frontier Foundation - Verizon Injecting Perma-Cookies to Track Mobile Customers, Bypassing Privacy Controls. https://www.eff.org/deeplinks/2014/11/verizon-x-uidh.

[4] ISPs Removing Their Customers' Email Encryption. https://www.eff.org/deeplinks/2014/11/starttls-downgrade-attacks.

[5] Official Gmail Blog - Staying at the forefront of email security and reliability. http://gmailblog.blogspot.com/2014/03/staying-at-forefront-of-email-security.html.

[6] The Official Microsoft Blog - Protecting customer data from government snooping. http://blogs.microsoft.com/blog/2013/12/04/protecting-customer-data-from-government-snooping/.

[7] The Washington Post - NSA infiltrates links to Yahoo, Google data centers worldwide. https://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html.

[8] The Washington Post - NSA uses Google cookies to pinpoint targets for hacking. https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/.

[9] Tor Meek. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[10] Tracking the FREAK Attack. https://freakattack.com/.

[11] CVE-2014-0160, 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[12] M. Akhoondi, C. Yu, and H. V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012.

[13] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*. ACM, 2001.

[14] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, S. Zanella-Beguelin, J.-K. Zinzindohoue, and B. Beurdouche. FREAK: Factoring RSA Export Keys, 2015. https://www.smacktls.com/#freak.

[15] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004.

[16] L. Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions of Networking*, 9(6):733–745, Dec. 2001.

[17] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference*. ACM, 2004.

[18] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design & Implementation*. USENIX Association, 2004.

[19] D. S. Miller, 2012. http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=89aef8921bfbac22f00e04f8450f6e447db13e42.

[20] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and Mitigating AS-level Adversaries against Tor. In *Proceedigns of the Network and Distributed System Security Conference*. Internet Society, 2016.

[21] N. Perlroth. NYTimes - China Is Said to Use Powerful New Weapon to Censor Internet, 2015. http://www.nytimes.com/2015/04/11/technology/china-is-said-to-use-powerful-new-weapon-to-censor-internet.html.

[22] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting In-flight Page Changes with Web Tripwires. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008.

[23] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed Internet Routing and Transport. *IEEE Micro*, 19(1):50–59, Jan. 1999.

[24] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2010.