Tetherway: A Framework for Tethering Camouflage

Steffen Schulz
Ruhr-University Bochum
Macquarie University
Technische Universität Darmstadt (CASED)
Bochum, Germany
steffen.schulz@rub.de

Ahmad-Reza Sadeghi, Maria Zhdanova
Technische Universität Darmstadt (CASED)
Fraunhofer SIT
Darmstadt, Germany
{ahmad.sadeghi, maria.zhdanova}@cased.de

Hossen A. Mustafa, Wenyuan Xu
University of South Carolina
Columbia, USA
{mustafah, wyxu}@cse.sc.edu

Vijay Varadharajan
Macquarie University
Sydney, Australia
vijay@science.mq.edu.au

ABSTRACT

The rapidly increasing data usage and overload in mobile broadband networks has driven mobile network providers to actively detect and bill customers who *tether* tablets and laptops to their mobile phone for mobile Internet access. However, users may not be willing to pay additional fees only because they use their bandwidth differently, and may consider tethering detection as violation of their privacy. Furthermore, accurate tethering detection is becoming harder for providers as many modern smartphones are under full control of the user, running customized, complex software and applications similar to desktop systems.

In this work, we analyze the network characteristics available to network providers to detect tethering customers. We present and categorize possible detection mechanisms and derive cost factors based on how well the approach scales with large customer bases. For those characteristics that appear most reasonable and practical to deploy by large providers, we present elimination or obfuscation mechanisms and substantiate our design with a prototype Android App.

Categories and Subject Descriptors

C.2.0 [**Security and Protection (e.g., firewalls)**]

Keywords

Mobile Networks, Tethering Detection, Traffic Obfuscation

1. INTRODUCTION

The success of smartphones is having a tremendous impact on the usage and development of mobile phone networks. On the customer side, low prices and the ability to run sophisticated applications result in a perpetual use of Internet services, such as email, video streaming and social networks. Ubiquitous Internet connectivity enables business professionals to access company resources while traveling, transforming idle time in airports, trains, and hotels into office hours. Since local WiFi connections in hotels or airports are often expensive and sometimes even unavailable, customers are tempted to "tether" their laptops and tablets to the mobile network connection provided by their smartphone. Already, many commodity smartphones such as Android phones or Apple iPhones have integrated mechanisms for sharing the network connection, or can be conveniently modified to do so.

On the provider side, the rapid growth of data usage on devices like tablets and smartphones has incurred large investments to optimize and reduce traffic load on the network infrastructure. Systems and methods are being developed to dynamically optimize hosting locations of content, broadcast data in 3G networks and to adjust the bit rate of content streams based on type and network load. In this setting, unexpected network usage can induce significant network overhead, breaking network optimizations and cost calculations. As such, tethering imposes a significant burden on mobile communication networks, a cost factor that providers like to accommodate in appropriate data plans.

Thus, tethering is usually prohibited for private contracts in most countries, and providers started only recently to offer data plans that explicitly allow tethering at extra cost[1]. In Europe, providers also generally require separate data plans for tethering, and in a few cases providers deployed large-scale tethering detection[2]. However, a limited free riding behavior has been accepted by providers for many years and today, many users do not see why they should pay more

WiSec'12, April 16–18, 2012, Tucson, Arizona, USA.
Copyright 2012 ACM 978-1-4503-1265-3/12/04 ...$10.00.

[1] For example, the AT&T "unlimited" data plan does not include tethering. Several customers, suspected of tethering, were informed to be switched to a different data plan, unless they report back to AT&T committing to stop their tethering [16]. Similarly, Verizon recently started to detect and redirect tethering users to their website, asking them to switch to a tethering data plan [19].

[2] For instance, O2 was reported to contact tethering customers via phone, to "alert" them to the terms and conditions, and reserve the right to disconnect them [30].

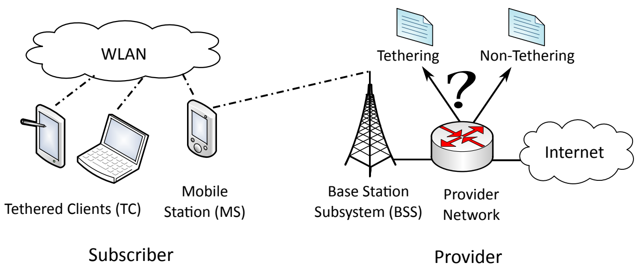

Figure 1: Tethering Detection Scenario

only because they share their paid bandwidth with other devices. Hence, the recent enforcement of contractual agreements through tethering detection is met with resistance, and many users attempt to hide their tethering usage [11].

Regardless of whether tethering should cost extra or not, the fundamental question for customers and providers alike is how to detect tethering, and at what cost. Naturally, there are many possibilities for detecting, e.g., the use of "untypical" web browsers by inspecting application layer content, or performing a generic traffic pattern analysis. However, these mechanisms can be rather costly to apply to large customer bases, and subscribers may apply obfuscation techniques.

To understand the issues associated with tethering, we must identify and analyze the potential mechanisms for the provider to classify traffic as tethering or non-tethering, approximate the costs for these mechanisms and investigate the effort for the subscriber to defeat classification.

Contributions and Outline.

In this work, we provide the first overall analysis on the mechanisms and feasibility of detecting tethering, and investigate how hard it is for the client to hide from such detection mechanism. After introducing the general problem of tethering detection in Section 2, we classify possible detection methods in Section 3 and assign cost factors to them based on the respective associated effort or cost for the provider. In Section 4, we discuss how the vast majority of practical tethering detection mechanisms can be defeated, verifying the feasibility of our approach with an Android-based prototype implementation. After reviewing related work in Section 5 we conclude in Section 6.

2. PROBLEM DESCRIPTION AND MODEL

We consider the tethering detection scenario as illustrated in Figure 1: The subscriber uses a mobile station (MS) to connect to the mobile broadband network of the provider. The MS is a highly customizable smartphone that may be used by the subscriber to connect additional *tethered clients* (TC), to share the mobile network connection. The provider aims to prevent such usage by classifying the network traffic of its subscribers (customer base) into either tethering or non-tethering traffic. The undesired tethering traffic can then be blocked or directly billed according to the provider's policy. On the other hand, the subscriber aims to hide its use of tethering, i.e., to confuse or circumvent the tethering detection of the provider.

Note that there are also other cases of "tethering" where the subscriber shares the mobile connection of the provider not only between own devices but also with third parties. Alternatively, more powerful mobile devices may (soon) be running virtual machines, creating a "virtual" tethering system. From the perspective of the provider, it is hard to distinguish these types of tethering from each other. However, since all of these approaches result in the same network setup and incur similar load on the mobile provider's network, we assume that they are equally undesirable.

The subscriber could also deploy a VPN tunnel to hide connection details from the provider. However, VPNs introduce configuration and compatibility issues, and increase network delay due to overlay routing. VPNs are also easy to detect, so that a widespread use of VPN to hide tethering may result in a ban of VPNs for cheap data plans.

2.1 Adversary Model

For the purpose of this investigation we treat the mobile provider as the adversary. We assume that the provider has full control over the network connection of the subscriber and may re-route, insert, modify or block transmitted packets. Additionally, the provider is able to read all transmitted data, including application layer information, except in cases where the subscriber uses end-to-end encryption mechanisms like SSL or VPN.

However, the provider is subject to certain practical limitations: The subscriber has full control over the programs running on the mobile station MS and any tethered client TC. Moreover, manipulation of application layer content or active OS fingerprinting may be detected by the subscriber and regarded as an attack, which we consider costly for the provider. Some attacks also require more resources than others, such as application layer inspection or stateful tracking of connections, increasing the cost for tethering detection.

We model these limitations by considering the cost or practicality ("effort") of the attack with regard to the various criteria that may be relevant for the mobile provider. In particular, we rate the effort as "low", "medium", "high" for the following criteria:

Impact Type: We distinguish passive, active and destructive detection methods. Passive methods simply monitor transferred network traffic, such as TCP/IP source and destination header fields, and incur *low* effort. In contrast, active attacks manipulate the data that the provider transfers on behalf of the subscriber or inject custom packets to prompt a reaction from the subscriber. This generally requires more resources due to realtime traffic processing and tracking, which we rate as *medium* effort. Some active attacks can result in noticeable interruption of ongoing network communication for the subscriber and thus are not suitable for large-scale scanning of a provider's customer base. We consider such destructive attacks *impractical*.

Protocol Layer: We distinguish attacks on network layer (*low*), application layer (*medium*) and "behavior layer" (*high*). Application layer attacks are generally more costly than network layer attacks since more complex protocol parsing and interaction is required, while lower layer protocols can be processed by most common hardware. With "behavior layer" attacks we denote the collection of traffic meta-data, which can encompass simple characteristics like timing and size of packets or more complex connection patterns. Due to the required long-term observation of individual customers we rate this as *high* effort attack.

Privacy Violation: We distinguish attacks that are not privacy-critical (*low*) from those that work with privacy-sensitive data (*medium*) (e.g., inspection of application layer data) and attacks that modify or inject application data (*high*). The latter approaches are often problematic with regard to data protection laws, and especially the undesired modification of user data is strictly prohibited in many countries [18].

Pre-condition: We categorize attacks as either unconditional (*low*) or conditional (*medium*). Unconditional attacks can always be launched, e.g., traffic analysis or querying the MS for active fingerprinting. In contrast, conditional attacks such as special HTTP or DNS access patterns depend on the behavior of the targeted subscriber and may be more or less likely to occur. Note that some pre-conditions may be easily met by longer-term passive observation, while others are only realistic if the provider can actively manipulate the MS, such as sending uncommon IP packets for OS fingerprinting. If a pre-condition is unlikely to occur even in active attack scenarios we denote the attack as *impractical*.

Detection Effort: We categorize detection methods based on signatures (*low*), heuristics (*medium*) or profiles (*high*). Heuristic and profile-based methods suffer from increased costs due to the computational efforts required for traffic processing, and profile-based methods are the most expensive as they additionally need traffic profiles to classify collected traffic with high accuracy. For example, network layer fingerprinting uses a simple database lookup ("signature") to classify OS implementations and is generally cheaper than traffic classification with statistical analysis and machine learning.

Using this classification, we derive the overall effort for each attack as the maximum of the efforts for each particular criterion. For example, the effort of evaluating the HTTP *User Agent* header is *medium*: It is a low-impact attack (*low*) but the inspected data is on application layer and may be considered privacy-sensitive (*medium*).

2.2 Communication Architecture

We assume a standard 2G/3G network setup on the provider side and a regular TCP/IP LAN[3] on the subscriber side. The mobile station MS dials into the provider's network and provides multiple interfaces such as wireless LAN, Bluetooth and USB to connect to local devices of the subscriber.

2.2.1 Tethering Technology

The tethering mechanism that connects the subscriber's LAN to the provider's WAN can be implemented in several ways. Most commonly, the MS is used either as a modem or IP gateway. Historically, the modem solution was used as it requires the least resources on the MS. However, the IP gateway solution is preferred on todays smartphones as it allows the simultaneous use of voice and data services. Some applications also offer other tethering techniques like application layer proxies or port-forwarding, however, such solutions provide only limited connectivity and are not easily deployed in combination with, e.g., VPN or VoIP software. Hence, we focus on the case of tethering where the smartphone acts as an IP router and gateway for the LAN, forwarding IP packets between LAN and WAN.

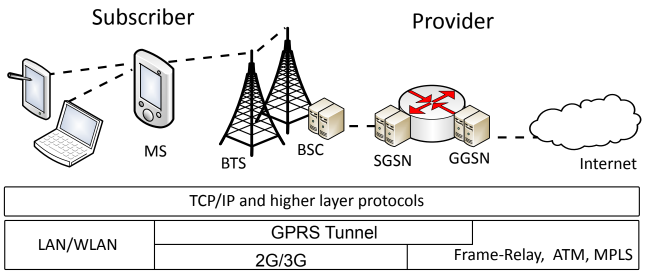

Figure 2: Overview of the 2G/3G communication architecture.

Technically, IP gateways for tethering on mobile phones are implemented using Network Address Translation (NAT), specifically Network Address Port Translation (NAPT) [33]. In NAT the port numbers and request identifiers of UDP, TCP, ICMP and other protocols are used to multiplex connections from the private LAN IP address space to the single, public IP that is typically issued to the mobile station MS by the provider. The deployment of NAT has three major consequences for tethering detection: (1) NAT transforms forwarded IP traffic, resulting in a modified traffic pattern that may be used to directly detect NAT; (2) a tethered client TC is not directly reachable from the WAN, so that, e.g., active fingerprinting by the provider will always only detect the MS itself but not TC; (3) since NAT is designed to be transparent, TCP and UDP payloads are transmitted unmodified, resulting in several options for tethering detection at application layer.

2.2.2 Mobile Networks Architecture

Figure 2 shows an overview of the 2G/3G mobile communication architecture. On the left, the MS is connected to one or more tethered clients via, e.g., wireless LAN. The mobile station acts as an IP router with NAT for these LAN clients, forwarding their IP packets through the provider's 2G/3G network in a General Packet Radio Service (GPRS) tunnel. The provider's Base Transceiver Stations (BTSs) forwards the GPRS frames on link layer to the Base Station Controller (BSC) and then to the Serving GPRS Support Node (SGSN) [1]. Once in the provider's network, GPRS frames are separated from voice traffic and forwarded to the Gateway GPRS Support Node (GGSN), where they are finally forwarded to the Internet [2].

As can be seen in Figure 2, the lowest protocol layer that is transported all the way from the MS through the provider's network is the topmost IP layer. Any lower layer information in the subscriber's LAN/WLAN is discarded already at MS, where forwarded TCP/IP packets are encapsulated in GPRS frames in the same way as locally generated ("non-tethered") packets, and are thus indistinguishable on link layer. Although the IP layer information can have some effect on lower layers, such as frame length and timing, these characteristics are also extractable at the IP layer. Hence, we reduce our analysis of tethering detection mechanisms to IP and higher layer protocols used by MS and TC.

[3] Note that we use the term *TCP/IP* in this work to refer to the complete TCP/IP protocol stack with UDP and ICMP.

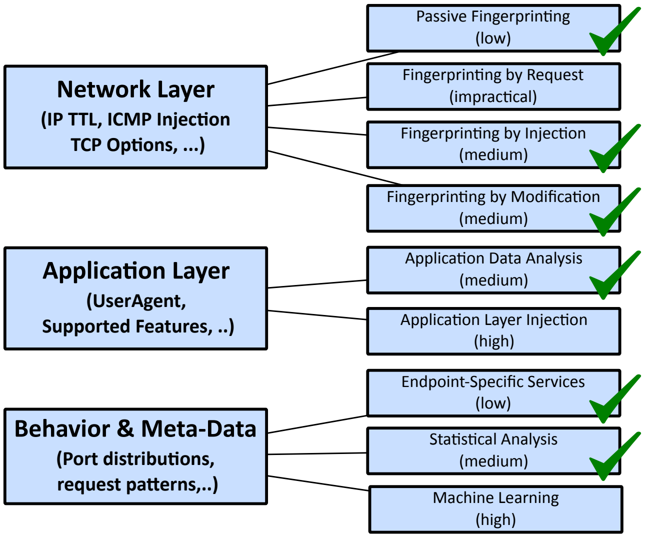

Figure 3: Classification of tethering detection mechanisms. Tetherway includes defenses against all *low* and *medium* effort detection techniques.

3. DETECTING TETHERING

Superficially, tethering detection appears similar to well-known mechanisms from network or OS fingerprinting and analysis of application behavior. However, most common fingerprinting techniques are not actually applicable and more complex attacks like active manipulation or traffic analysis quickly become too costly to be applied to the huge customer base of a mobile provider.

A general overview of the different types of attacks is provided in Figure 3. Abstractly, we can classify the possible tethering detection mechanisms into (1) network layer fingerprinting, (2) application layer inspection and manipulation, and (3) behavior and traffic pattern analysis.

In the following, we discuss each of these approaches in detail and rate their feasibility and practicality by assigning costs based on the adversary model in Section 2.1

3.1 Network Layer Attacks

Tethering detection methods on network layer can be generally described as fingerprinting attacks. However, we must emphasize that tethering detection is different from the well-known OS fingerprinting attacks: In OS fingerprinting, the adversary aims to detect the OS type running on a specific remote machine, while in tethering detection we are interested in identifying *additional* hosts *behind* the MS, or the fact that Network Address Translation (NAT) is enabled at the MS. Hence, many standard attacks are not effective while other and new attacks become useful. In the following we thus distinguish passive attacks and active attacks that are based on (i) requests, (ii) injection, and (iii) manipulation.

3.1.1 Passive Fingerprinting

Passive observation of network and transport layer header fields and traffic flows can be used to distinguish hosts behind NAT or directly detect the use of NAT. In the IP header, the fields for Differentiated Services (DS), Explicit Congestion Notification (ECN), IP Flags and especially Time To Live (TTL) may be used differently depending on the OS that the packet originates from (e.g., [5, 35, 25]). Similarly, information on the Initial TCP Window Size and Sequence Number, the supported types, values and order of TCP Option fields such as Maximum Segment Size (MSS), Window Scaling and Timestamps can be used to discriminate different TCP/IP implementations behind NAT [7, 37, 9]. We consider such passive network layer attacks as *low* effort, as they are the least invasive and most scalable.

3.1.2 Fingerprinting by Request

Most established OS fingerprinting techniques assume an active adversary to query the target with specially crafted TCP or ICMP requests [32, 4]. However, such requests are usually answered by the mobile station MS itself, and thus can only identify the MS and not the TCs. To the best of our knowledge there are also no active fingerprinting attacks that directly detect the use of NAT by querying the MS. Hence, active fingerprinting attacks based on requests are generally ineffective, as long as they only target the MS itself and cannot detect its use of NAT, so that we denote such attacks as *impractical*.

In the following we discuss two different approaches for active fingerprinting which either inject or manipulate packets of *existing* connections. The NAT at the MS keeps track of such existing connections and will forward injected or manipulated packets as long as they can still be recognized as part of a known ongoing connection[4].

3.1.3 Fingerprinting by Injection

An active fingerprinting attack can traverse the NAT barrier by injecting packets into an existing connection, so that they are recognized and translated by the NAT engine. For the UDP protocol, no such attacks are known in the related work, likely due to its inherent simplicity. TCP packet injection attacks are possible but problematic, as packet injections desynchronize the TCP connection between the original sender and receiver [44]. This requires the provider to constantly manipulate the TCP packet stream until an opportunity for re-synchronization occurs, or until the connection is ended, or otherwise results in a noticeable interruption of the TCP session of the subscriber. We rate the resulting longer-term realtime traffic manipulation as a *high* effort attack.

The remaining network layer protocol that is frequently used today is ICMP. For active fingerprinting of hosts behind NAT, an ICMP error message could be injected from the provider based on an existing UDP or TCP connection. A NAT engine is required to statefully rewrite and forward several types of such messages to ensure transparent IP operation [34]. However, ICMP errors often signal critical faults in the forwarding of IP packets and are thus not designed to generate an observable response. In fact, most ICMP errors will immediately terminate the respective UDP or TCP connection, leading to noticeable interruptions that make such approaches *impractical*.

We have identified only one error message that is (1) regularly forwarded into the LAN behind a NAT gateway, (2) handled by standard TCP/IP implementations, including An-

[4] In TCP and UDP, the connection is identified by the source and destination ports and IPs addresses. ICMP packets are recognized by their identifiers or, in case of ICMP error messages, based on the UDP/TCP port and IPs contained in their embedded TCP/UDP fragment [33, 34].

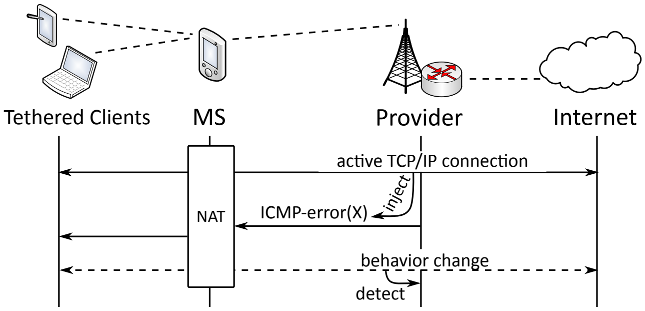

Figure 4: ICMP injection attack for detecting NAT.

droid, and (3) results in changes observable to the provider if handled by a tethering client: the ICMP "fragmentation needed and don't fragment bit was set" error message. As illustrated in Figure 4, the provider may inject such a message as a response to an ongoing TCP/IP connection. The error is forwarded by the NAT gateway, manipulating one of the tethered hosts to believe into a smaller Maximum Transmission Unit (MTU) for this particular *IP route*, i.e., for the IP layer connection between the source and destination of the respective TCP or UDP connection. As a result, the target of the injected ICMP error (TC or MS) will start transmitting smaller packets for that particular IP route, which can be observed by the provider. But since the ICMP error is regularly forwarded to only one of the potentially multiple hosts on subscriber side, the provider can then detect the use of tethering by checking if *all* other connections using the same IP route adopt the same reduced MTU.

The main drawback of this attack is that it only works if multiple IP connections to the same destination hosts are opened at the same time by different tethering hosts, as otherwise no difference in the behavior of any two connections can be observed. While this pre-condition cannot be easily induced by the provider, it is also not very unlikely to occur, especially when considering the high frequency at which email, instant messaging and news aggregation clients connect to popular Internet services today. We classify such ICMP injection as a *medium* effort attack due to its active manipulation of traffic and the required pre-condition.

3.1.4 Fingerprinting by Modification

The adversary may also manipulate IP and TCP headers that are destined for the MS and potential TCs with the goal to create observable changes in the connection state or behavior. The scenario is similar to the ICMP MTU attack illustrated in Figure 4: When manipulating the IP or TCP headers of an ongoing TCP/IP connection, the manipulation propagates through the NAT barrier at the MS. The tethered clients $TC_1, TC_2, \ldots$ may then act differently depending on the included options, creating a client-specific feedback that is observable by the provider.

One example of such an attack is to set the TTL for inbound packets towards the MS to "1". Such packets can be received by the MS, but an additional forwarding to the TCs would decrement the TTL to "0", so that the packet is dropped at the NAT gateway. Similarly, the Flags field or Header Checksum of the IP or TCP header may be manipulated to detect different types of operating systems through their different ways of error handling. However, all these manipulations involve a high risk to noticeably interrupt ongoing connections, making them *impractical* for scanning of large customer bases.

There are also several less common optional headers in IP and TCP that may be handled and supported differently by different TCP/IP implementations. The provider may exploit the difference in endpoint behavior to detect the existence of multiple hosts at the MS, by injecting TCP options into existing TCP/IP streams and observing the response behavior. Due to their optional nature, the injection or deletion of such options will usually not break the connection, however, they still require an active manipulation and stateful observation of the traffic flow, resulting in *medium* effort attack.

3.2 Application Layer Attacks

A large number of application layer characteristics can be used to differentiate hosts behind a NAT, either explicitly based on meta-data information in protocol headers or implicitly, by exploiting the different features supported by individual applications. However, the line between mobile and desktop "Apps" becomes increasingly blurred by the rapid progress in smartphones and particularly tablets, which are getting as complex and powerful as desktop systems.

3.2.1 Application Data Analysis

The easiest way to identify the number and type of hosts and applications on application layer is passive application layer inspection or Deep Packet Inspection (DPI). Well known examples for application layer data that identify systems and applications behind the MS are the *User Agent* field in the HTTP header and the host identifier strings sent as part of eMail or Instant Messaging (IM) communication [8]. Protocols with more complex negotiation of features and algorithms, such as TLS or IPsec, also often exchange an implementation-characteristic sets of supported features during their negotiation phase, which potentially allow to distinguish multiple hosts behind a NAT gateway even if the actual user data is encrypted.

Passive application layer attacks are highly practical. The required DPI can be done asynchronously and is already available for several policy-enforcement scenarios in the network management. Hence we can rate this approach as *medium* effort.

3.2.2 Application Layer Injection

For active attacks, i.e., the injection of code into transferred websites or the redirection of users to the provider's servers, the attack surface for distinguishing TCs is practically unlimited[5]. However, such active attacks are also rather resource intensive, have a high risk of getting noticed and may be interpreted by the client as intrusion and privacy invasion [19, 30]. According to our cost model, the effort for such attacks is therefore considered to be *high* due to potential privacy violations.

3.3 Traffic Metadata Analysis

The third main category of tethering detection concerns the analysis of traffic meta-data. Similar to application layer attacks, the area of statistical traffic analysis is very large [10]. However, we can single out two approaches that

[5] See, e.g., `http://browserspy.dk` and the demonstration at `https://panopticlick.eff.org/`

result in rather efficient detection mechanisms, while the more sophisticated traffic analysis with machine learning is more resource intensive.

We emphasize that the popular packet size and timing analysis usually requires long-term observation and sophisticated machine learning techniques to classify traffic with reasonable accuracy. As such it constitutes a rather *high* effort attack when applied to the large customer base of a mobile phone provider.

3.3.1 Endpoint-Specific Services

Many operating systems and applications can be identified based on the individual Internet services they use. For example, Android phones are unlikely to connect to the Microsoft Windows update servers or to package repositories provided by most large Linux distributions. Instead, they will mainly connect to the Android Market for software update information, and to the configured Google, Exchange or Facebook accounts for synchronization of contact data, etc. Similar considerations apply to individual applications such as anti-virus scanners, office suites, PDF viewers and Java runtime environments that are known to regularly contact the servers of their respective vendors.

A special case of this category are tethering applications provided by the mobile provider itself, which explicitly signal the use of tethering by switching the provider's access point (APN). These applications are regularly shipped with iPhones, but also with many Android phones.

Endpoint-specific tethering detection requires only passive traffic monitoring on network layer, without any particular post-processing of data. While the considered events (preconditions) are usually not easily to trigger by the provider, they are still likely to occur. For example, many systems check for software updates as soon as an Internet connection is established. Hence we rate this kind of attacks as *low* effort.

3.3.2 Statistical Analysis

With this category we denote approaches that use simple frequency or random distribution analysis, i.e., which employ rather simple analysis models [6, 13]. Well-known examples of this category concern the random distribution or range of values in individual network layer header fields, such as the IP ID fields [35, 5] or the clock skew in values of the TCP timestamp header [9]. We also found that the port-multiplexing of NAT leads to distinctive changes in the source port range and distribution. In contrast to previous works [35], we found this characteristic to be highly suitable for tethering detection (cf. Section 4.2.1 and 4.3).

A common property of these attacks is that they concern only lower-layer protocols and require only short to medium observation time, so we consider their overall cost as *medium*.

3.3.3 Machine Learning

Apart from the direct evaluation of header fields and connection states, a major approach in the classification of traffic is the traffic pattern analysis with machine learning. This approach is characterized by collection and model-based analysis of network meta-data such as traffic volume, packet size and timing, as well as more complex patterns like the number of simultaneous TCP connections for given destinations, or with specific higher-layer protocols.

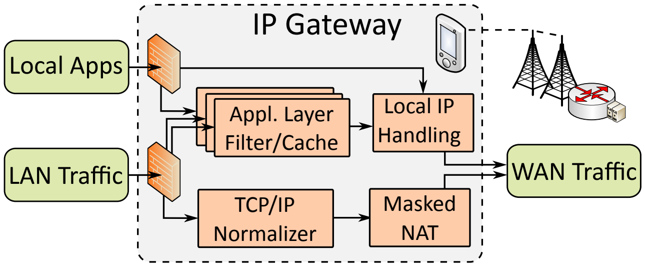

Figure 5: Tethering Normalization Architecture.

An example attack that could be mounted in this way to reveal tethering is the detection of multiple browser caches at the client, which results in different HTTP object request patterns. Alternatively, the provider may attempt to distinguish particular TCP flow control algorithms implemented at the MS, based on how individual TCP connections increase or throttle their packet rates over time.

The data processing phase involves the use of generalized profiles that are created by observing and then either implicitly or explicitly classifying large amounts of data (supervised or unsupervised learning, see [28, 42]). However, supervised learning involves a high effort in the learning phase, and the results of unsupervised learning require similarly high effort to confirm potential matches and filter false positives [20]. In the end, all types of learning-based traffic analysis appear to require long-term observation, careful system analysis and post-processing. Due to the high costs in regard to Detection Effort and Protocol Layer we rate their overall effort as *high*.

4. DEFEATING TETHERING DETECTION

To evaluate the practicality of tethering detection, we have developed a generic architecture for normalization and obfuscation of the tethering characteristics identified in Section 3. In this context, normalization describes the process of modifying the distribution of values for the previously identified characteristics, such that they are close or identical to the distribution of a non-tethering system. We have implemented a prototype to defeat against the most practical (i.e., cheapest) attacks identified above and compare its performance against simple VPN solutions.

4.1 Tethering Normalization Architecture

Figure 5 depicts our approach to normalization and obfuscation of tethering characteristics in network traffic. There are two main components for traffic normalization: (1) the Application Layer Filter and Cache and (2) the TCP/IP Normalizer with Masked NAT. A packet filter is used to assign data streams to the respective components, and after processing all packets are forwarded through the same WAN interface to the provider, as discussed in Section 2.2.

The Application Layer Filters must be developed specifically for each individual application layer protocol, like HTTP, and can thus also normalize application-specific patterns such as the *User Agent* meta-information in HTTP headers. Moreover, they can block, aggregate and cache queries to obfuscate access behavior and traffic patterns or imitate desired patterns. After the application layer payloads are processed, they are forwarded using the local IP handler of the MS, so that no TCP/IP layer normalization is required.

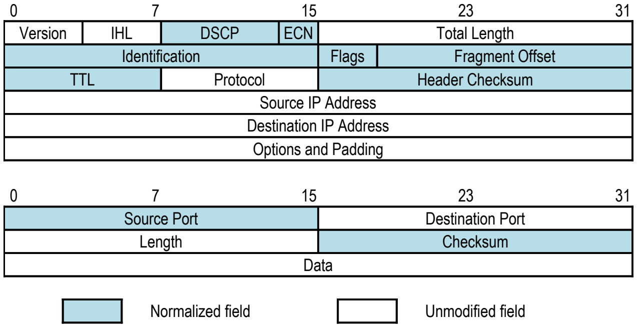

Figure 6: Normalized IP and UDP header fields.

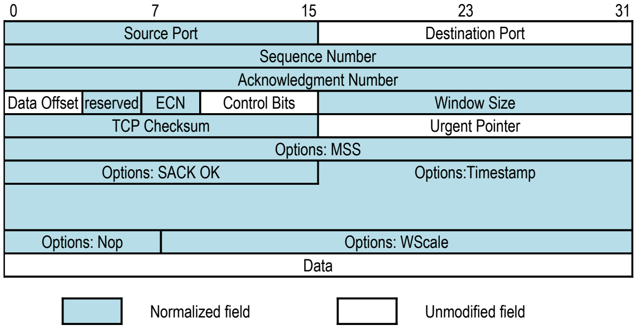

Figure 7: Normalized TCP fields and options.

The main disadvantage of this approach is that it must be implemented for each individual application protocol and cannot handle unknown protocols.

Hence, we have added the TCP/IP Normalizer as a generic network layer normalizer in case no Application Layer Filter was defined for a particular data stream. This also includes many protocols where application-layer processing is not possible or not deemed necessary, such as encrypted traffic. As a network layer normalizer, its capabilities are limited to the normalization of header fields and filtering of unusual requests and header options, as detailed in Section 4.2.1. The normalization is followed by the Masked NAT component, which translates the IP address range of the LAN traffic to that of the WAN traffic [33]. The NAT is masked in the sense that the range and distribution of the modified header fields are indistinguishable from that of the local IP stack (cf. Section 4.2.1).

4.2 Tethering with Tetherway

To verify the practicality of network normalization on smartphones, we implemented an Android App for tethering normalization system called *Tetherway*. Tetherway is based on the popular Android App *android-wifi-tether*[6], which can use the ad-hoc wireless network, USB or Bluetooth personal network of the MS to connect to tethered clients, providing a standard IP gateway with DHCP and NAT. In the following we describe the implementation details of the TCP/IP Normalizer, Masked NAT, and two Application Layer Filters for DNS and HTTP. We denote packets that are destined to the Internet or provider network as *outbound* packets, and packets destined for the MS or TCs as *inbound* packets.

4.2.1 TCP/IP Normalization

We have implemented a TCP/IP packet normalization using the *libnetfilter-queue* extension[7] of the Linux firewall subsystem. The extension allows us to program custom packet filters in the Linux userspace, enabling arbitrary rewrite of network packets to simulate non-tethering behavior.

IP Header Normalization.

On IP layer we reset the DS field for outbound packets and enabled ECN. We set the TTL field to 64 and also reset the IP flags and fragment offset to disable fragmentation, as this is the default behavior in Android smartphones.

Note that while NAT cannot handle fragments, Linux transparently defragments packets before NAT processing so that they are treated in the same way, regardless of whether they are destined for the MS or TCs.

More involved is the adjustment of the IP ID field to elicit the same random distribution as the IP ID values of a standard Android platform *without* NAT, i.e., a randomized initial value that is incremented with each packet of the same TCP/IP session. To imitate this behavior of modern Linux kernels we have implemented a corresponding stateful rewrite of IP IDs for TCP/IP sessions, using local copies of the respective randomization functions in the Linux kernel. For UDP, no normalization is required since the standard behavior to set the IP ID to zero is the same as it is done by regular NAT in Linux. Only in case of DNS requests, the IP ID may contain a random number to mitigate DNS cache poisoning attacks [21]. However, we have deployed a caching DNS proxy for this case. Finally, our system filters all inbound and outbound IP options as they are usually not used or needed.

ICMP MTU Exceeded Injection.

The ICMP injection attack we proposed in Section 3.1.3 can be partly mitigated by replicating ICMP errors on the NAT gateway: Similar to the approach of IPsec, the gateway can record received ICMP MTU errors from the WAN and distribute them not only to the respective LAN host referenced in the error message, but also to all other hosts that send a packet larger than the reported MTU to the same destination IP. As a result, the provider will not receive any packets larger than the MTU size previously injected. The provider could still wait for the reduced MTU values to time out on the individual TCs, an event that occurs at different times depending on the deployed OS at the TCs. However, this event occurs only after 1-2 minutes, complicating detection as many TCP connections are not sufficiently long-lived.

Hence, while our defense is not perfect, the attack cost is increased by requiring long-term observation and the precondition of multiple long-lived TCP connections is less likely to occur, so we rate the new effort for this attack as *high*.

TCP and UDP Header Normalization.

Figure 6 and 7 give an overview of the normalized fields in the typical TCP/IP headers. For the TCP header, we normalize the Sequence Number field and the ECN and Reserved Flags, as well as several TCP option headers. The Ac-

[6] `http://code.google.com/p/android-wifi-tether/`

[7] `http://netfilter.org/projects/libnetfilter_queue/`

knowledgment Number, Window Size and Checksum fields are also updated as a consequence of other corrections.

Just like in the IP header, the Explicit Congestion Notification (ECN) flags can be safely reset to always enable ECN support: ECN is supported by all modern hosts, and systems that do not support ECN suffer the resulting performance penalty in any case.

To normalize the TCP (Initial) Sequence Number (ISN), we statefully track TCP connections and use the functions for randomized ISN selection from the Linux kernel to select a new ISN for each TCP connection over NAT. Since the endpoints of a TCP connection rely on the sequence numbering for ordering and acknowledging received TCP segments, we then record the *offset* between the original and newly chosen ISN for each new connection and adjust (1) all subsequent Sequence Numbers on outbound and (2) all Acknowledgment Numbers on inbound packets accordingly. Hence the provider is unable to distinguish ISNs for tethering, and all Sequence Numbers are correctly rewritten regardless of the packet reordering.

TCP Option Headers.

We must normalize the various TCP Option headers to mitigate the problem of fingerprinting by modification (cf. Section 3.1.4). For this purpose, we purge all TCP Options that are not used by Android from inbound as well as outbound packets, i.e., all Option headers except for the Message Segment Size (MSS), Selective ACK (SACK), Timestamp (TS), Window Scale (Wscale) and No Operation (NOP) fields. For outbound packets, we furthermore normalize the order and content of the remaining supported TCP Options.

The MSS Option can be normalized using the iptables *clamp-mss-to-pmtu* option. The SACK and Wscale Options are used only in the initial TCP handshake to signal support for selective acknowledgments and window scaling. The SACK Option does not contain any actual values and thus does not require normalization, except for its location within the TCP header. However, the Wscale Option on mobile stations often advertises the smaller Window Scale factor than the one used for the LAN and WLAN interfaces of desktop systems. To transparently rewrite this value, we statefully track the TCP connection, recording the original Window Scale factor before reseting it to the typical Wscale factor of “1” for Android. For all outbound TCP headers, we recompute (left shift) the Window Size header value to compensate for the smaller Window Scale factor. In the worst case, this modification *reduces* the absolute receive window assumed by the sender, possibly reducing TCP performance. However, this is also the expected behavior for a regular (non-tethering) MS. Finally, the TS Option adds two timestamps to all TCP packets to let endpoints compute the precise Round-Trip Time (RTT) of a connection [17, 9]. To make the TS values indistinguishable from the timestamps of the MS, we simply replace all timestamps inserted by the TCs with values generated by the MS. The resulting change in the computed RTT is negligible, as TC and MS are typically very close.

Apart from the SACK Option, all of the discussed Options could be synthesized on the gateway without breaking the TCP session. However, currently we treat a missing TCP Option as an error and do not currently implement such synthesis.

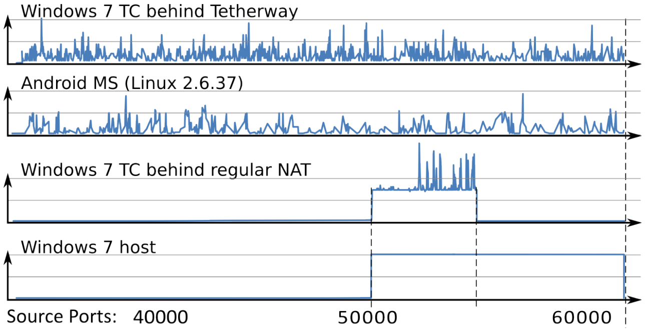

Figure 8: Source port distributions for Android and Windows 7 without NAT, with NAT and with Tetherway.

Masked NAT.

The source port multiplexing by NAT, or more specifically Network Address Port Translation (NAPT), can lead to characteristic source port distributions: Firstly, the standard Linux NAPT implementation uses a different port range than the standard range for ephemeral source ports in outbound connections. Moreover, Linux NAT tries to preserve the original source ports whenever possible, so that source port distributions of different TCs are likely to remain visible regardless of NAT (cf. Figure 8).

Fortunately, the Linux NAPT subsystem also supports two parameters to set the port range and enable port randomization. By source code inspection we confirmed that enabling port randomization disables the preservation of source ports, and that the employed randomization function is the same as the one used in the ephemeral source port selection for local traffic. Hence, we can eliminate differences in the distribution of source port values sent via NAT versus that of local connections by the MS, by enabling randomized port mapping and setting the same range of ports as used by the regular ephemeral port selection of Android.

4.2.2 *Application Layer Proxies*

We have implemented application layer filtering for the two most used services, DNS and HTTP. In both cases we have deployed standard application layer proxies to implement the filtering and caching required for eluding tethering detection.

For HTTP, we deployed *Privoxy*[8] as a filtering proxy. Privoxy is used to anonymize HTTP traffic for users of *Tor*[9] and provides extensive and well-tested rules for filtering, e.g., user tracking based on cookies and Webbugs, or embedded active content such as Adobe Flash. To obfuscate object request patterns on the behavior layer, the filtering proxy is backed by a caching parent proxy, *Polipo*[10]. As a result of this construction, the HTTP request behavior becomes similar to that of an endpoint with only a single browser cache, as it would be expected from a non-tethering MS.

For DNS, we use *dnsmasq*[11] as a local DNS cache on the MS. We implement a simple DNS filter by returning the local host IP address 127.0.0.1 for blacklisted DNS records.

[8] http://www.privoxy.org/

[9] http://www.torproject.org

[10] http://www.pps.jussieu.fr/~jch/software/polipo/

[11] http://thekelleys.org.uk/dnsmasq/doc.html

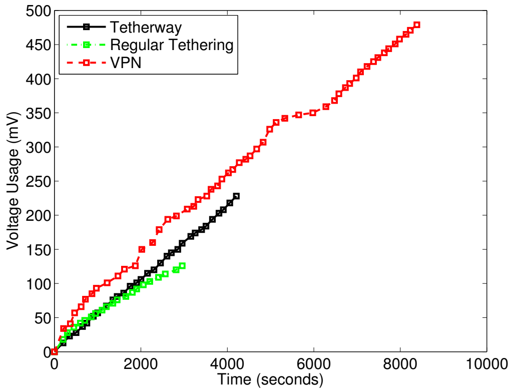

Figure 9: Power consumption for Tetherway compared to regular tethering and VPN.

This can be used to block behavioral patterns like accesses to Windows Update or the standard Windows time synchronization servers.

4.3 Evaluation

We evaluate the efficiency of our normalization engine by comparing the phone's power consumption while downloading 960 objects (50MB total) from 2 websites using (1) regular non-normalizing tethering, (2) a typical VPN software and (3) our Tetherway prototype. In particular, we compared Tetherway against the android-wifi-tether App that it is based on, and against Juniper Junos Pulse VPN for Android. As can be seen in Figure 9, the total energy consumption and the amount of time of Tetherway for completing the same task are close to that of regular tethering and notably below that of a VPN client. This confirms our subjective impression that responsiveness of the VPN connection was notably lower, likely due to the additional hops introduced by the VPN tunnel.

To confirm the proper normalization of header fields, we compare the distributions of critical header fields such as the source ports in Figure 8 and TCP initial sequence numbers in Figure 10. In each case, a clear difference in the distributions of standard Android and a Windows 7 TC can be identified. Specifically, Figure 8 shows a highly predictable port usage for Windows 7 hosts, while Android uses a different port range with randomized source port selection. Similarly, the random distribution of the TCP ISN values shown in Figure 10 is distinctively different for Windows 7 and Linux, with Windows hosts using a larger range of values that are not changed by regular NAT. In contrast, the distribution of the Windows 7 client behind Tetherway is similar to the one expected from a non-tethering Android MS.

We conclude that our normalization is efficient and effective, making tethering detection much harder for the provider. VPN software can also be used to hide all of the identified tethering characteristics, by setting up the MS as a VPN gateway for the TCs. However, VPNs are not designed for this purpose, resulting in noticeable network overhead and increased power consumption. Moreover, with a widespread use of tethering via VPN, mobile providers may be tempted to restrict the use of VPNs to specialized "business" data plans, for exactly the same reasons that they try to restrict the use of tethering today.

4.4 Limitations

Tetherway focuses on the most common and most easily detected tethering characteristics, as illustrated in Figure 3. Several possibilities remain to identify tethering setups. Our HTTP and DNS caches do not enforce a particular packet timing or obfuscate patterns in the network access behavior of email or IM clients. However, such normalization can be added, e.g., using the Linux network emulator *netem*[12].

Furthermore, although HTTP and DNS traffic accounts for a large amount of device mobile traffic, many other protocols such as Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP) or the several Instant Messaging protocols are also in widespread use. However, most modern applications employ encryption based on TLS, hiding any application layer characteristics.

Note that several more complex detection mechanisms, e.g., based on traffic analysis, may still be viable if the provider applies some preliminary filtering to reduce the number of suspects, e.g., based on the amount of traffic usage. However, an in-depth analysis of these approaches is outside the scope of this work.

5. RELATED WORK

To our knowledge, this work is the first to consider the problem of tethering detection, which can be seen as a generalization of the previously considered NAT detection. We systematize tethering detection methods and show that the most cost-effective techniques can be mitigated efficiently.

Vendors like Cisco and Sandvine already provide tethering detection solutions [12, 31]. However, they currently inspect only a rather simple mix of network and application layer headers, indicating that injection or traffic analysis attacks are indeed more costly. Consequently, first tethering Apps also provide correspondingly simple detection countermeasures like resetting the IP TTL field[13] or using proxies[14].

Regarding the general problem of tethering detection, we identify three major categories of related work: (1) Revealing multiple hosts or different operating systems behind an IP gateway, (2) detecting use of NAT and (3) performing general statistical analysis of network traffic.

Direct NAT detection. The use of NAT and the counting of hosts behind NAT was previously considered as a general problem of network mapping and measurement. For example, it was proposed to detect NAT based on analysis of the IP ID field in the IP header and to count "NATed hosts" through reconstruction of IP ID sequences [5]. In another approach, a naive Bayesian classifier was used to detect hosts behind NAT based on IP TTL, DF, initial TCP window size and TCP SYN packet size [7]. Similarly, the authors of [35] propose to combine multiple parameters to increase detection accuracy, using data such as IP ID, TTL and source port distribution.

We consider these attacks among the most practical since they involve only simple statistical analysis on network layer data. We extend on these works by showing that source port distribution is a highly practical tool for detecting NAT, and that countermeasures for all previously presented network layer NAT detection techniques are feasible even on resource-limited devices (cf. Section 4.2.1).

[12] See, e.g., `tcn.hypert.net/tcmanual.pdf`

[13] PdaNet 5.01 `http://junefabrics.com/index.php`

[14] `androinica.com/tag/clockwordmod-tether/`

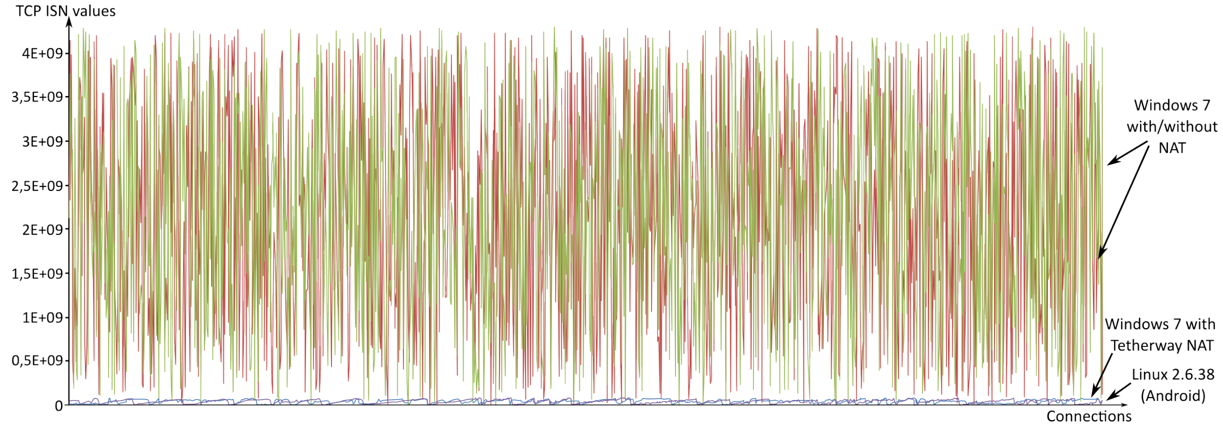

Figure 10: Random distribution of TCP Initial Sequence Number (ISN) values for Windows, Windows behind regular NAT, Linux and Windows behind Tetherway (imitating Linux ISNs)

Fingerprinting and Scrubbing. OS fingerprinting is a general technique to identify remote systems by the difference in their TCP/IP stack implementations [3, 14, 24] and network scanners like Nmap[15] or p0f[16] are widely available.

However, as shown in our analysis, the most common active fingerprinting attacks, which work by sending specially crafted requests to the target, are ineffective in case of tethering detection (cf. Section 3.1). Similarly, most fingerprinting attacks that rely on the modification of header fields often break the affected connection, making them unsuitable for tethering detection by providers.

Multiple works propose normalization or scrubbing of network traffic, such as IP Personality[17], Morph [39] or ipMorph [29]. Network-based normalization has also been proposed to normalize traffic for processing in network intrusion detection systems [26, 15]. The main difference between *Tetherway* and existing solutions is the purpose of normalization, which determines the kind of distributions to imitate and in our case requires, e.g., the use of masked NAT, while defense against active fingerprinting by request is not needed.

Traffic Analysis. Traffic analysis has been used to detect network applications [36], online activities [43], behavior profiles of client systems [42] and properties of the encrypted network, namely routing and flows [13]. Furthermore, it was shown that traffic analysis could yield results even if the transferred packets are encrypted or timing is masked [13, 41, 22, 6]. On the other hand, various countermeasures against traffic analysis have been proposed, ranging from network layer meta-data normalization [38, 40, 27] to efficient application layer obfuscation [23].

We believe that complex traffic analysis attacks are not practical for tethering detection with large customer bases. Instead, we only normalize header fields such as the TCP/IP source ports, which elicit highly characteristic patterns in case of tethering. While we deploy application layer proxies for simple obfuscation of packet timing and connection patterns, we defer a detailed analysis of possible novel attacks and countermeasures for later work.

6. CONCLUSION

We have presented the first general analysis and classification of tethering detection techniques. Our analysis indicates that tethering detection is a heuristic, highly fragile process, and many techniques are easily defeated by a modified mobile station.

VPN software can be used as a readily available tool to evade tethering detection, but reduces network performance notably and is easily detected by the mobile provider. In contrast, our Tetherway prototype closely imitates regular non-tethering traffic with modest overhead.

The large amount of potential attacks makes comprehensive precautionary measures impractical. However, given a known specific detection mechanism, additional normalization or obfuscation mechanisms can often be deployed. As a result, we expect that any widespread tethering detection method will quickly be countered through community effort, increasing the development cost at the provider until tethering detection itself is not cost-effective anymore.

For future work, it would be interesting to further investigate the potential and efficiency of traffic analysis in face of basic defenses such as proxies or random packet delays.

7. REFERENCES

[1] 3rd Generation Partnership Project (3GPP). *General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface*, 2011.

[2] 3rd Generation Partnership Project (3GPP). *Mobile Station (MS) - Serving GPRS Support Node (SGSN); Subnetwork Dependent Convergence Protocol (SNDCP)*, 2011.

[3] J. M. Allen. OS and application fingerprinting techniques, 2007.

[4] O. Arkin. ICMP Usage in Scanning. Black Hat Briefings, 2000.

[5] S. M. Bellovin. A technique for counting NATted hosts. In *SIGCOMM Workshop on Internet measurment (IMW)*. ACM, 2002.

[6] L. Bernaille and R. Teixeira. Early Recognition of Encrypted Applications. In *PAM*. Springer, 2007.

[7] R. Beverly. A robust classifier for passive TCP/IP

[15] `http://nmap.org/`

[16] `http://lcamtuf.coredump.cx/p0f.shtml`

[17] `http://ippersonality.sourceforge.net/`

fingerprinting. In *Passive and Active Network Measurement*. Springer, 2004.

[8] J. Bi, L. Zhao, and M. Zhang. Application presence fingerprinting for NAT-aware router. In *Knowledge-Based Intelligent Information and Engineering Systems*. Springer, 2006.

[9] E. Bursztein. Time has something to tell us about network address translation, 2007.

[10] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok. A Survey on Internet Traffic Identification. *Communications Surveys & Tutorials, IEEE*, 11(3):37–52, 2009.

[11] B. X. Chen. AT&T tells free tethering customers it's time to pay up. Wired, 2011.

[12] Cisco Systems. *ASR 5000 Series Enhanced Charging Services Administration Guide Addendum*, 2011.

[13] D. Cousins, C. Partridge, K. Bongiovanni, A. W. Jackson, R. Krishnan, T. Saxena, and W. T. Strayer. Understanding encrypted networks through signal and systems analysis of traffic timing. In *Aerospace Conference*. IEEE, 2003.

[14] F. Gagnon and B. Esfandiari. A hybrid approach to operating system discovery based on diagnosis. *Int. J. Network Mgmt.*, 21(2):106–119, 2011.

[15] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security*, 2001.

[16] P. Horowitz. AT&T cracking down on unofficial iPhone tethering & MyWi users. OSXDaily, 2011.

[17] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, 1992.

[18] W. John, S. Tafvelin, and T. Olovsson. Passive Internet Measurement: Overview and Guidelines based on Experiences. *Computer Communications, 33 (5)*, 2010.

[19] C. Johnston. Verizon blocks unlicensed tethering, insists it can charge extra. ArsTechnica, 2011.

[20] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *CoNEXT Conference*. ACM, 2008.

[21] A. Klein. OpenBSD DNS cache poisoning and multiple O/S predictable IP ID vulnerability, 2007.

[22] L. Lu, E.-C. Chang, and M. Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2010.

[23] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 2011.

[24] G. F. Lyon. Remote os detection via TCP/IP stack fingerprinting. `nmap.org/book/osdetect.html/`, 2011.

[25] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Passive and Active Measurement Conference (PAM 2010)*. Springer, 2010.

[26] G. R. Malan, D. Watson, F. Jahanian, and P. Howell. Transport and application protocol scrubbing. In *INFOCOM*, 2000.

[27] R. E. Newman, I. S. Moskowitz, P. Syverson, and A. Serjantov. Metrics for traffic analysis prevention. In *Workshop on Privacy Enhancing Technologies (PET)*. Springer, 2003.

[28] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008.

[29] G. Prigent, F. Vichot, and F. Harrouet. IpMorph: fingerprinting spoofing unification. *Journal in Computer Virology*, pages 329–342, 2010.

[30] P. Rasmussen. O2 to crack down on iPhone tethering cheats. Fierce Wireless, 2009.

[31] Sandvine. Sandvine tethered device detection solution and service revenue enhancement. Case Study, 2011.

[32] R. Spangler. Analysis of remote active operating system fingerprinting tools. http://www.packetwatch.net, 2003.

[33] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, 2001.

[34] P. Srisuresh, B. Ford, S. Sivakumar, and S. Guha. NAT Behavioral Requirements for ICMP. RFC 5508, 2009.

[35] K. Straka and G. Manes. Passive detection of nat routers and client counting. In *Advances in Digital Forensics II*. Springer, 2006.

[36] G. Szabo, D. Orincsay, B. P. Gero, S. Gyori, and T. Borsos. Traffic analysis of mobile broadband networks. In *Wireless Internet (WICON)*, 2007.

[37] G. Taleck. Ambiguity resolution via passive OS fingerprinting. In *Recent Advances in Intrusion Detection*. Springer, 2003.

[38] B. R. Venkatraman and R. E. Newman-Wolfe. Transmission schedules to prevent traffic analysis. In *Annual Computer Security Applications Conference (ACSAC)*. IEEE, 1994.

[39] K. Wang. Frustrating OS fingerprinting with Morph. Talk at the fifth HOPE conference, 2004.

[40] C. V. Wright, S. E. Coull, and F. Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Network and Distributed Systems Security (NDSS)*. Internet Society, 2009.

[41] C. V. Wright, F. Monrose, and G. M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *Journal of Machine Learning Research*, 7:2745–2769, 2006.

[42] K. Xu, Z.-l. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: Behavior models and applications. In *Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*. ACM, 2005.

[43] F. Zhang, W. He, X. Liu, and P. G. Bridges. Inferring users' online activities through traffic analysis. In *Wireless network Security (WiSec)*. ACM, 2011.

[44] O. Zheng, J. Poon, and K. Beznosov. Application-based TCP hijacking. In *European Workshop on System Security (EUROSEC)*. ACM, 2009.