# Protecting Privacy with Protocol Stack Virtualization

Janne Lindqvist
Helsinki University of Technology, Finland
Dept. of Computer Science and Engineering
janne.lindqvist@tml.hut.fi

Juha-Matti Tapio
Helsinki University of Technology, Finland
Dept. of Computer Science and Engineering
jmtapio@verkkotelakka.net

## ABSTRACT

Previously proposed host-based privacy protection mechanisms use pseudorandom or disposable identifiers on some or all layers of the protocol stack. These approaches either require changes to all hosts participating in the communication or do not provide privacy for the whole protocol stack or the system. Building on previous work, we propose a relatively simple approach: *protocol stack virtualization*. The key idea is to provide isolation for traffic sent to the network. The granularity of the isolation can be, for example, flow or process based. With process based granularity, every application uses a distinct identifier space on all layers of the protocol stack. This approach does not need any infrastructure support from the network and requires only minor changes to the single host that implements the privacy protection mechanism. To show that no changes to typical applications are required, we implemented the protocol stack virtualization as a user space daemon and tested it with various legacy applications.

## Categories and Subject Descriptors

C.2.5 [**Computer-Communication Networks**]: Local and Wide Area Networks—*Internet*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

## General Terms

Security, Design, Experimentation

## Keywords

privacy, protocol stack virtualization, pseudonymity

## 1 Introduction

In this paper, we consider the privacy of mobile computer users. Often, the most effective way to protect one's privacy is to remain anonymous, that is, not to reveal one's name or other identifiers at all. When identifiers are absolutely necessary, such as in maintaining communications sessions and seamless mobility, pseudonyms can be used instead. This paper presents a technique for improving pseudonymity of mobile Internet hosts: *protocol stack virtualization*.

Quite a bit of research has been done on anonymizing Internet communication [3, 6, 8, 10, 17, 23, 25, 32, 34]. The protection mechanisms vary greatly depending on the location and capabilities of the assumed adversary. There are, however, some common themes. One is that the defender avoids giving out identifiers [3, 6, 25, 32], either explicit or implicit, which could be used to recognize it and to trace its actions over time and over different locations. The attacker, on the other hand, tries to use all available clues, such as forgotten identifier fields, protocol metadata, payload data and statistical profiling, to link together multiple appearances and actions of an individual user or computer. Another common theme [8, 10, 17, 23, 34] is that the defender hides among other users, that is, blends into a crowd. Indeed, anonymity is often measured as the size of the set of users among which one is indistinguishable (the anonymity set). This makes it, by definition, impossible to achieve anonymity alone.

Based on the above, it becomes clear that the complexity of modern software will cause problems for privacy protection. The multitude of networking protocols and network-enabled applications found on a typical computer will make it easy to fingerprint the computer based on its individual software configuration and overall communication patterns, even if each protocol and application in itself has been carefully anonymized. Additionally, there will be conflicting requirements from the various protocols regarding the lifetime of pseudonyms and when they are replaced. Yet another issue is that wireless network interfaces can be continuously enabled, rather than switched on sporadically by the user, to meet the needs of the various applications. This may increase the number of occasions when there are not enough mobile hosts at the same network to provide the safety of a crowd.

Our main idea is to create a seemingly separate instance of the protocol stack for each application. The increased isolation between applications creates two kinds of benefits. First, each application will have a distinct set of identifiers for the link, network and transport layers. This enables us to deploy previously proposed privacy-protection mechanisms, such as address randomization, on per-application basis, or even with more fine-grained granularity, without trying to configure one protocol stack that meets the peculiar require-


*WPES'08,* October 27, 2008, Alexandria, Virginia, USA.

ments of all the applications. Second, the unique identity of the user (or computer) will be split so that it appears as a group of unrelated anonymous or pseudonymous entities, each of which is typically involved only in a single communication session with a single peer. This makes some of the profiling attacks more difficult to implement because, in order to profile the user or computer, the attacker needs to first establish which sessions belong to the same computer.

We have implemented the approach as a host-based network address translator (NAT) for Linux, and tested it with a number of different real applications and low-end network infrastructure elements to show the feasibility and deployability of the approach. In the implementation, we have synthesized ideas from virtual NAT [38], a proposal to create a new IP address for every TCP connection [6] and transient addressing (TARP) [15].

The rest of the paper is organized as follows. In the next section we give background on privacy of mobile computers. Next, we state the problem setting and give considerations for the system design. Section 4 describes the design and implementation details and results of testing the implementation are given in section 5. We contrast our work to related work in section 6. We discuss the limitations and benefits of our approach in section 7, and outline further work in section 8. We conclude the paper in section 9.

## 2 Privacy of Mobile Computers

First, we briefly describe how information on different layers of the protocol stack can be used to establish privacy violations, depending on the location of the attacker. We start with the means available to a local adversary. Note that these means may not identify the user directly, but can neverthless be used to pinpoint distinct users with help of out-of-band means. The list is not exhaustive, but exemplifies the many ways a host and consequently the user can be identified. Starting from the lower layers of the protocol stack, the adversary has the possibility to fingerprint the devices with analog signals e.g. in the Ethernet [14] or e.g. by the WLAN device driver behavior [12]. The MAC layer and especially the MAC address can be used to identify the host [18]. Even implicit identifiers such as broadcast packet sizes and SSIDs of the wireless networks that the device has attached to can be used to identify the user [31]. And numerous explicit identifiers are leaked from enterprise domain controlled Windows laptops, such as, name server lookups, LDAP queries or printer configurations [4].

Based on the attack vectors described above, we observe that against local adversaries, the use of anonymous identifiers in upper level protocols is not enough for location privacy or identity protection. The host needs to invoke only one application that uses a public identifier and the anonymity of the anonymous protocol can be violated. For example, privacy extensions [29] for IPv6 stateless address autoconfiguration [39] or SIP [35] privacy service [32] do not help against a local adversary, if it can correlate the traffic with the MAC address or the identifiers in the application protocols.

The following means are applicable also to adversaries that are not local, that is, not in the range of the wireless or in the wired local area network, but for example, one router hop away. The IP address, TCP sequence numbers, IPsec ESP SPIs [3], or even the clock skew of TCP or ICMP time stamps [24, 27] can be used to identify the host. Also, if the authentication and key exchange protocol is not designed carefully, the use of encryption techniques can also be harmful for privacy [1, 2, 4]. And even dynamic DNS can be used to identify and track the user remotely [19]. Finally, the data that applications transmit can be used for correlation purposes. As a concrete example, only the login process is usually encrypted with Web based email services, and even though TLS-protected IMAP is common, encrypted connections for outgoing SMTP are not. Additionally, many Web based service providers such as online book stores, provide clear-text information about the identity of the user for a third party to observe.

## 3 Problem Definition and Design Considerations

We aim to minimize the information leaks of the user by isolating different traffic flows from each other. This protects the privacy of the user to a degree; even when the attackers find out that a particular user is in the network, they might not be able to easily deduce what is the affiliation or organization of the user.

To provide concrete examples of these information leaks, we refer to our previous work on chattering laptops [4]. The DHCP client can reveal the FQDN for the host in the home enterprise network. DNS and Netbios queries can reveal the host name. Application meta-data such as Web based email can reveal the username and the realname of the user. These information leaks are harmful both for location privacy and identity protection. The user can be tracked and identified when changing locations, and cannot remain even pseudonymous.

Based on above, we have taken a pessimistic view on the privacy of mobile computers. Our position is that eventually the user or the host operating system does something that will reveal the identity of the user and we desire to limit the consequences of this unavoidable circumstance. In other words the problem we address in this paper, is preventing linkability of user actions, operating system and the applications. Even though some of the network-enabled applications may take privacy into account, even *when* one application reveals the identity of the user, other network traffic originating from the same host can be linked to the user. Even though the other traffic may be encrypted (and in many cases it is not), a third party can gain useful information if it can know the identities of the communication parties even if the attacker cannot see the content of the communication.

We assume that the attacker can reside either in the local wireless or wired network or further away (one or more hops). The main focus of our work is to protect from a local attacker, for example, in a WLAN network, but at the same time, the approach protects the user from more distant attackers, too. The goal of the attacker is first to identify the user as distinctively as possible. The second goal is to gather other information related to the habits and network usage, for example, where does the user work, who he or she contacts.

A privacy protection mechanism should be very easy to deploy and use, because users might not be able to make rational decisions regarding their privacy. Thus, we need to minimize the disruptions to the user experience. To help

deployability, we do not assume any infrastructure from the network to participate in the privacy protection mechanism. Also, for deployability reasons, we do not want to introduce a burden to the peers that a host communicates with, since many service providers do not have incentives to protect the privacy of the user. Thus, the solution space is limited to approaches that provide unlinkability by modifying only a single host, namely the host that wishes to protect the privacy of the user.

Finally, one problem that is overlooked in the design of many privacy architectures is the desire to remain *reachable*. Although users appreciate the fact that their privacy is protected, they also desire to be reachable for example by their friends. These conflicting requirements need also be addressed by a practical privacy protection architecture. We summarize in the following lists the requirements and assumptions for our approach.

- To provide *deployability* we
  - do *not assume* any infrastructure.
  - change *only* a *single operating system*.
- The approach should not change the *user experience*, for example, we
  - do not require *decision making*
  - or *learning* from the user.
- The approach should allow the users to remain *reachable*.
  - e.g. allow inbound connections (e.g. servers)

# 4 Design and Implementation

The basis of the design is to use virtual randomized MAC addresses and new IP addresses for every flow or application. What the granularity of virtualization should be is beyond the scope of this work. The basis of the system already protects a lot regarding casual observers on the local link, however, we need more complexity to cover trivial fingerprinting attacks and avoid breaking *interdependent* protocols, such as ICMP error messages, or violating privacy with DNS lookups. The system consists of the following subsystems:

- MAC address randomizer
- IP address autoconfiguration
- system introspection
- protocol helpers
- inbound packet filtering

For every new MAC address used, the system must obtain a new IP address. With IPv4, we can use DHCP, and with IPv6, the stateless address autoconfiguration [39] or DHCPv6. Although we later argue that given the scarcity of IPv4 address space the approach is more practical for wide deployment with IPv6, we implemented the protocol stack virtualization for IPv4 in order to test it with a number of legacy applications available for IPv4. Nevertheless, based on our preliminary IPv6 implementation work we do not perceive that extending the work for IPv6 could introduce problems with IPv6 legacy applications.

System introspection and protocol helpers are needed to guarantee the functionality of some legacy applications. For example, ICMP error messages need to be mapped to the connections that are the reason for the messages. Further, to constrain the use of the address space, we need to map multiple different connections to a single application. We also need to filter some inbound connections to mitigate active attacks such as port scans that could reveal the use of protocol stack virtualization.

## 4.1 Inbound connections

If we accept all inbound packets for all the addresses that we have in use, the adversary performing a port scan of the IP address pool would get identical results for all the addresses, and thus, we could lose unlinkability. We can protect the unlinkability of the addresses if we accept incoming connections on only one address. Thus, we can drop inbound TCP and UDP packets that are not associated with an existing connection if they are not destined to our default source address. The limitation here is that all protocols that use inbound connections need protocol helpers or they have to use the one inbound address. Examples of inbound connections that could reduce the unlinkability would be incoming SIP or HTTP connections.

## 4.2 Methods for configuring addresses

There are three basic methods for obtaining an IP address. The first method is using addresses within a manually configured address block. This is mostly suited for testing as manually configured networks are not very typical and this method may require good understanding of the local network topology.

The second method is obtaining each address with DHCP. This method is mostly limited to IPv4 and there are some limitations. Querying DHCP causes some delays, in addition to the server overhead, because the DHCP needs at least two roundtrips. Also due to the scarcity of IPv4 addresses, in many cases the DHCP servers give out only a fairly limited number of addresses. Therefore with DHCP we have to focus on using only a minimal amount of addresses and minimizing the lifetime of each address.

There also is an additional risk associated with DHCP addresses. Some servers give out addresses sequantially and there is little entropy if there is only one virtualized user in the netblock amongst multiple non-virtualized users. In an otherwise quiet network all the virtualized addresses could be sequential. However, in many networks DHCP is the only practical way to obtain an address.

The third address selection mechanism is IPv6 stateless autoconfiguration [39]. A 64-bit prefix is picked from the Router Advertisements on the network and the host part of the address can be pseudorandom with the privacy extensions [29]. As long as the host listens to neighbor discovery messages to avoid using the same address as some other host, this mechanism provides a fairly fast and simple way to obtain new addresses. New IPv6 addresses can be obtained without any server involvement and it is enough to implement neighbor discovery protocol so that the router and other hosts can send packets to the virtualized address. This mechanism also avoids the entropy problem of the DHCP alternative because the host controls the used addresses.

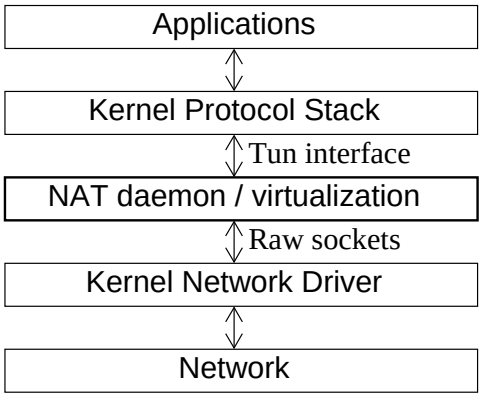


**Figure 1: Top-level implementation architecture**

### 4.3 Special considerations for DNS

Where regular data connections can be fairly reliably separately grouped, DNS traffic is more problematic. DNS requests reveal information about what the user is doing. For example, if a query of a website's IP address and a query of a domain's mail handler can be linked, the user's privacy may be compromised regardless of how well the actual data connections have been separated. Therefore, it is important to map DNS queries correctly together with the connections that they are related to. This presents a number of challenges for the implementation.

The DNS queries are separate from the applications in that they do not share server addresses or ports with the applications that caused them, and they might not even be sent by the application itself to the network. For example, in some configurations all DNS queries are sent by the operating system's name service cache daemon. Thus, we have to look into the queries and try to deduce mappings from there.

If there are no intermediate cache daemons between the applications and this introspection process, we can observe which application is making each request. Then we do not associate two applications with the same source address just because they happen to both query, for example, www.tkk.fi.

Regular local DNS caching is potentially a problem for virtualization. If caching is done, it should be implemented in such a fashion that each cached record would only be used with one associated source address. Otherwise it might be possible to observe the DNS queries sent by some address and see if it seems to already know some DNS records (especially NS-records) that have recently been queried by some other addresses. For example, assume a scenario where the user is accessing two different web sites, for example, the employer's site and some politically incorrect site. Someone controlling the employer's site (or able to modify the traffic from the site) and being able to listen to network traffic could then serve the client a web page linking in some invisible way (a small image, stylesheet or iframe) to the other site. If the linking caused the client to query the DNS for the politically incorrect site, the attacker could expect that the user had not visited the site with any virtualized IP address. If, on the other hand, the client did not have to query DNS, but knew the address of the other site already, it would help to prove that the client has recently accessed the site with some IP address.

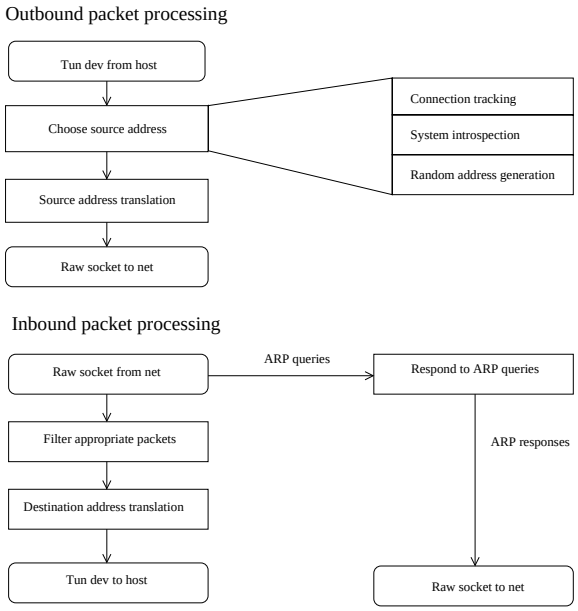


**Figure 2: Packet flow within the NAT daemon**

We considered implementing the DNS handler as a DNS proxy that the applications would have been configured to use. However, we decided it is better to implement it as a transparent proxy within the NAT daemon. This way we need less system configuration changes and the whole flow processing can be done within one piece of software. UDP packet introspection is a simple procedure and the process is tightly coupled with the other connection tracking mechanisms. Supporting TCP introspection for TCP-based queries is a more complex process, but that support can be combined mostly with protocol scrubbing. Thus, we do not perceive that implementing transparent handling for DNS over TCP would be problematic.

### 4.4 Implementation details

We implemented the protocol stack virtualization as a host-based network address translator (NAT) that instead of traditional NATs [37] provides a set of distinct identifiers for every network flow originating from the host. The implementation is a proof-of-concept daemon for Linux. In other words, every instance of an application is given a new IP address and virtual MAC address, and flow identifiers in transport protocols.

The host-based NAT daemon is logically placed between the operating system's regular protocol stack and the network driver as shown in Figure 1. To simplify prototyping, we implemented the daemon with Python in user space. The daemon uses a tun-device to communicate with the protocol stack and raw sockets to communicate with the network driver.

The flow of the data packets within the daemon is described in Figure 2. All inbound and outbound packets are basically routed through the daemon. To pass packets for the protocol stack virtualization code in the daemon, a special tun-device is set up. The tun-device is configured with a special IP-address that is visible only in the operating system and the system's default route is set to that interface. Thus, the kernel will by default route outgoing data packets

to that interface and pass them to our user space process. The NAT daemon reads those packets from the device and deconstructs them to modify the headers.

To map the outgoing packet to a new source IP address, we consider the IP protocol of the packet. For packets not associated with an already known connection we choose randomly the source IP address from an available pool and an unique MAC address for that IP address.

**Connection tracking** Connection tracking is required for TCP and various other protocols and therefore the NAT daemon needs to use the same source address for packets related to the connection. A connection is mapped to a specific source address with a key formed by inspecting the packet. For TCP we use a triple ('tcp', destination address, destination port) as the key to identify a connection. We considered including source port in the key, but we observed that web browsers typically use multiple TCP connections from various source ports to download pages and images and using multiple source addresses for downloading content from one web server would be wasteful.

Because UDP applications typically use one UDP port to communicate with multiple destinations, there is no need to associate multiple local ports with one connection. Thus, we consider it reasonable to use a 4-tuple ('udp', destination address, source port, destination port) as the key for UDP connections instead of the triple that we use for TCP.

**Protocol interdependencies** For ICMP we use by default only a tuple of ('icmp', destination address) as the key, but this does not cover all possible cases. ICMP error messages have to be inspected and associated with the connection that caused them. Otherwise messages such as *port prohibited*, *time exceeded* and *MTU discovery* would be sent from the local host to peers using random source address. This would cause the peer to discard the packets and needlessly reveal the use of virtualization. All other protocols will be mapped to a default source address so that unsupported protocols will continue to work, assuming they can survive NAT.

**System introspection** In addition to the connection categorization desribed above, we improve connection tracking and grouping by two configurable mechanisms: system introspection and protocol helpers. For TCP and UDP it is possible to look up which local user id or application caused a specific connection. We call this process system introspection because we analyze the Linux kernel network structures published in the `/proc`-filesystem. By scanning `/proc/net/tcp` and `/proc/net/udp` we can produce the user id and inode of the socket that matches the packet being analyzed. To find which applications have the particular socket open, we scan the `/proc/<pid>/fd`-directories to look for links to the open socket. For example, if we have Skype running, we can find out if a certain TCP or UDP packet belongs to a connection used by the Skype process. Depending on configuration, we use this information to group all Skype data packets into one logical connection and only use one source address for all Skype packets.

**Protocol helpers** We can also augment connection tracking by protocol helpers that detect and group multi-flow connections such as active FTP or SIP. The helpers can be used in the same manner as typical NAT helper modules. They can both detect multiple related TCP and UDP flows and perform packet rewriting in cases where the local application reports the locally visible IP address to the peers.

Our prototype runs in user space and it therefore introduces some extra latency. To mitigate the latency, the implementation sends gratuitous ARP replies before the outbound data packets for the IP and MAC addresses we use. These ARP packets may cause some routers to remember the MAC and IP addresses for a while and thus avoid needing to wait for extra ARP queries. Not all routers accept these messages and we do not expect them to be generally useful for in-kernel or realtime implementations. The gratuitous ARP might introduce a possibility to fingerprint hosts that are using protocol stack virtualization, but this is only an implementation issue, not inherent to the proposed approach.

The NAT daemon performs a straightforward network address translation on the outbound packets and replaces the source address using the method described above. In addition to modifying the source address in the IP header, the IP checksum has to be recalculated. Because the TCP and UDP headers use the IP source address in their checksums, those have to be recalculated as well. The modified packet is sent to the next router via a raw socket using the chosen virtual MAC address as the hardware source address.

The physical network interface needs to be set to promiscuous mode so that the network interface card will also listen to packets that are destined to the virtual MAC addresses. The NAT daemon reads captured inbound packets from a raw socket. The packet is further processed if it is either an ARP request for one of the virtualized IP addresses, or destined to one of the virtualized addresses. For packets recognized as belonging to some existing connection the daemon performs reverse translation for the destination IP address mapping it back to the special internal address.

## 5 Results

In this section, we give the results on testing our implementation on real applications and show that even low-end switches are not a hindrance for deploying the implemented system.

**Application tests** We tested the NAT daemon with various Linux applications to evaluate the feasibility of the idea. Tests were run in a typical home ethernet environment. The client running Linux had a 2 GHz AMD Athlon CPU and a VIA Rhine network interface card. The traffic was routed through an A-Link ADSL router. Generated traffic was captured and further analyzed.

We chose *ping, traceroute, ftp, Firefox web browser, Skype, OpenVPN* and *VideoLAN* clients as test applications to provide variety. Ping and traceroute performed as expected, communication with each end host was done with a different source address.

As was to be expected, *ftp* did not work until passive mode was enabled. In active mode ftp asks the remote end to open a connection to the local IP address, but ftp reports the internal interface address which does not work. It should, however, be noted that this type of problem is not specific to protocol stack virtualization and exists with all protocols that open multiple connections to addresses reported by the hosts themselves. As with all NAT-systems, this problem can be fixed with a special protocol helper module. However, we do not perceive it as a problem that active FTP does not

work without a protocol helper module, because e.g. most Web browsers default to passive FTP today.

*Firefox* worked well. In our tests we saw no differences to regular browsing. Nevertheless, a balance needs to be chosen with the connection tracking. Web browsing can open multiple outbound connections to the same server. If these connections are all virtualized separately, addresses can be wasted. However, if all web requests to a specific server are made from the same source address, unlinkability might suffer if the user visits multiple independent web sites that are hosted on the same destination IP address.

*Skype* is an interesting case due to its P2P architecture. It seems to work well on top of protocol stack virtualization. In our tests Skype was very conservative in its use of UDP ports, but it did use a lot of TCP connections to various nodes around the world. In an example test, Skype quickly used approximately 15 virtualized addresses.

*OpenVPN* is a TLS-based VPN implementation that runs on top of either TCP or UDP. It worked well on top of both transport types. A live TV simulcast with a 300-400 kbps MMSH-feed also worked flawlessly with the *VideoLAN* client. We noticed nothing unusual with these applications running on top of protocol stack virtualization.

Overall in testing sessions when not using the introspection, the tested applications created approximately 34 TCP connections, which were mapped to 19 IP addresses. The biggest consumer of IP addresses in our testing was Skype. Browsing one site with a browser typically uses a single IP address, a few more if some content is fetched from separate servers. As for UDP connections, there were respectively 87 conversations between UDP endpoints (IP address and port). Almost all of them were Skype related. These UDP connections used 55 virtualized IP addresses.

We emphasize that these connection counts were from tests without any protocol helpers and without using introspection to group connections on an application basis. Therefore, it is possible to significantly reduce address usage by using a configuration where some applications use only one address. We estimate that some amount of unlinkability would be realistic with even a few public IPv4 addresses obtained, for example, by DHCP.

The most computationally demanding operation in the NAT daemon is the optional local system introspection to find out the user id or the application. Therefore, we decided to test it separately. On the same above mentioned desktop system we tested how long it takes to lookup the responsible user id or application. The system had 117 processes running and the tested process was near the end of the process list. Thus, the figures approximately represent the worse case scenario regarding the lookups on similar desktop configurations. We performed the tests with Python's timeit-framework with 1000 rounds. For user id lookup the average lookup time was 0.5 microseconds and for user id and application the lookup took on average 14 milliseconds. We estimate that the additional delay caused by the introspection lookups is low enough for them to be useful. We would like to note that due to connection tracking, it should be enough to do the lookups only for the first packets of each connection. This can further be optimized by for example skipping application lookup for certain well known ports such as 80 (http).

Based on these results we estimate that a simple user session consisting of browsing a few sites and using some basic network applications (P2P applications excluded) would need tens of virtualized IP addresses with protocol stack virtualization. Use of P2P applications would raise the need for addresses significantly unless the protocol-specific helper modules or introspection are used to aggregate together all the connections belonging to one specific application.

Additionally we measured the extra latency caused by the user space implementation by comparing plain ICMP ECHO "pings" within one Ethernet segment with and without our protocol stack virtualization implementation. With virtualization we measured an average round trip time of 1 ms (100 packets) while without virtualization we measured an average of 0.2 ms. This test did not include any introspection lookups. Thus we had approximately one millisecond of extra delay caused by our implementation. We consider this delay reasonable and not prohibitive with regard to good scalability, and we believe that an in-kernel implementation would remove the introduced delay.

**Low-end switch test** We tested the feasibility of using multiple virtualized hosts in a switched network. Potentially, careless use of MAC and IP addresses could result in problems. We implemented a separate program that rapidly sends packets with different MAC addresses, and a responder echoes these packets. A third machine attached to the same switch was used to observe the network, for example, when packets are broadcasted.

To find out if the virtualization has negative effects on switching performance, we tested a commercial off-the-shelf low-end WLAN access point. The device started gradually broadcasting some of the packets after reaching a hundred entries in the arp table. At approximately 4000 entries, everything was broadcasted. Packet loss, however, did not occur in the test setting.

Another interesting observation was that the default arp cache size in Linux 2.6.15 kernel is 1020. When the limit is exceeded, the `sendto` system call reports an ENOBUF error, and prevents sending of more UDP datagrams. The limit can naturally be increased.

These simple measurements already show that the virtualization approach does not affect the accessibility of the access network, even though there would be multiple clients using the approach at the same time with coarse granularity of virtualization.

## 6 Related Work

Although we have named our architecture protocol stack virtualization, it is different from full virtual machine monitors such as VMware [40], Terra [13], Crossbow [11] or even from network virtualization optimizations [26] provided by Xen [7]. These environments do provide the possibility to virtualize the whole protocol stack, but their design purpose is not unlinkability of network traffic, instead they provide the possibility to virtualize full operating systems for execution environment isolation. Additionally, the computational and memory overhead of our solution compared to a full virtualization environment is significantly less, and our approach does not require anything new for the user to learn. PlanetLab [33] allocates resources (slices) on a network of hosts spread around the world. Every application deployed in the PlanetLab requires a slice which corresponds a collection of virtual machines on different hosts around the world. The stack virtualization approach we have proposed shares

the concept in a sense, however, we provide a virtual instantiation of the protocol stack for applications in a single operating system.

Previous approaches [3, 25] that do not rely on infrastructure for location and identity privacy have proposed to use pseudorandom or disposable identifiers for all layers of the protocol stack. However, in these approaches the identifier spaces are shared by the applications, and therefore the traffic can be linked to the host by observing the applications. Similar observations can be made from approaches that change only the MAC address [18] or rely on the network to assign pseudorandom MAC address [22]. However, some of the approaches [3, 25] take into account maintaining end-to-end connectivity when changing network attachment, but require changes to both communicating parties. We have, however, left this kind of mobility management beyond the scope of our work.

Sender and receiver anonymity, and thus, identity privacy, can be achieved with many approaches that rely on an infrastructure such as Chaum's MIXes [8], onion routing [17], Crowds [34], the second generation onion router (Tor) [10], or with information slicing on unreliable overlays [23]. These approaches do provide unlinkability of the traffic that is routed through the infrastructure. However, if every flow originating from the host is not routed through the infrastructure, the user is vulnerable to correlation attacks at least in the local network. We do consider these approaches complementary in the sense that if the user uses, for example, Tor, to provide end-to-end sender and receiver anonymity, with our approach, the unprotected traffic cannot be linked to the Tor traffic. We further compare our approach to Tor in the discussion.

The leaks that we attempt to mitigate with protocol stack virtualization can be considered to be a part of information flow control problem space. There are theoretical models that consider information flow properties such as noninterfence [16]. In the information flow control field, there are also practical models and their implementations, for example, on how mutually distrusting can express their privacy policies and the system enforces the requirements simultaneously [28]. "Privacy as an operating system service" proposes that the operating system includes a privacy module for scrubbing sensitive data [21]. As a concrete example, the operating system should automatically remove data such as names and social security numbers, in addition to meta-data and "para-data" such as the author's username from sensitive Word documents. However, in our approach, we are concerned with identifying information on all layers of the protocol stack that could compromise the identity and location privacy of the user. In our view, these approaches are complementary, and we agree that a complete privacy solution should actually take into consideration also the (meta|para)data that is sent to the network by the applications. Similarly to above, TightLip [42] provides a way to handle access control decisions when user has defined what is sensitive data and who is trusted. For every process handling sensitive data, the TightLip system launches a doppelganger process. The operating system follows these processes, and if they make system calls with different arguments, it is concluded that the original process might be divulging sensitive data. Privacy as an operating system service and TightLip both need an administration to configure the privacy policies for the system. Also, another Doppelganger [36] provides a user-friendly way to manage cookies. The approach protects from useless use of privacy-harming cookies, but does not help with the personalization problem of the online web stores mentioned in Section 2. Further, the approach is limited only to Web cookies.

In "Privacy, Control and Internet Mobility" [6], the authors make a short note that for privacy protection purposes in mobility or roaming solutions such as Mobile IP, applications should be more aware of mobility and the protocol stack be more aware of application privacy requirements. However, it does not suffice that applications are aware of what happens in the operating system or the protocol stack since they are not aware of the privacy requirements of the system. As mentioned in the introduction, the authors also propose that with IPv6, every new TCP connection that is initiated from the host should use a new IP address. The authors have not implemented the approach, and do not consider local attackers; their idea was to provide equivalent privacy for IPv6 that NATs provide for IPv4.

VNAT [38] provides process migration with connection virtualization. The idea is that the IP addresses used for the first connection a process establishes to another process on the network remain as the fixed IP address pair for the connection. When the process is moved to another host, the connection can be recovered with an update mechanism. VNAT provides address translation for these virtual addresses, however, the approach requires changes to both peers, is not designed with privacy in mind, and even though the authors use the term "physical address" to denote the current IP address of the host to add, they do not virtualize the MAC addresses.

Finally, TARP [15] proposes to use multiple addresses per host for improved firewalling. The authors also use the virtual machine methaphor: "each client process group conceptually runs in a virtual machine, with an independent IP address space" [15]. However, TARP does not virtualize the MAC address and its purpose is to simplify state-keeping in firewalls, not protecting the privacy of the user.

# 7 Discussion

In this section, we first discuss the limitations of our approach and then proceed to contrast it to Tor, and to the design considerations given in Section 3.

## 7.1 Limitations

The first three limitations presented can be roughly categorized as the boundaries of current legacy systems and the last four limitations present different possibilities for attackers to circumvent the privacy protection architecture. These attacks are not, however, only applicable to our approach, previously proposed privacy protection mechanisms are also vulnerable to similar attacks.

**Medium Constraints** The approach we have presented is best suited for broadcast media such as WLAN, and in practice, open WLAN networks. In a broadcast medium, if e.g. the device driver cannot be fingerprinted, a third party cannot deduce who is sending the traffic with the virtualized set of identifiers. However, this is not the case for example with switched Ethernet. If the attackers have access to the switch, they can see from which port all the traffic with the different identifiers are coming from. Nevertheless, with many network switches, if there is plenty of port activity, it

is not trivial to deduce the ports where traffic is originating. Similarly, if the user is connected at home to a single ISP, the possibilities to shield against the ISP with the approach are limited.

**MAC-address authentication** Many legacy authentication systems use MAC addresses to authenticate the mobile computers attached to the network. Even though the approach by itself is insecure, because the addresses can be easily spoofed, it is widespread. The protocol stack virtualization approach cannot be used as presented in these networks. The approach could be modified, to use other MAC addresses that are allowed to the network, by passively observing and recording the addresses. However, we do not propose this approach to be used, because in some countries, bypassing authentication systems by forging addresses might be illegal. A viable alternative is to implement a backwards compatibility option. In networks, where MAC address based authentication is used, we would only provide multiple IP addresses to the applications. And, if that is not possible, the implementation could just behave as an unmodified operating system would, but notify the user of the reduced privacy level.

**Limits of the address space** Both IPv4 and Ethernet MAC addresses limit the applicability of the approach. Even though we implemented the approach with IPv4 to be able to test it with real applications, we consider IPv6 with stateless address autoconfiguration [39] to be the real use case for the approach. The network interface identifiers of autoconfigured IPv6 addresses can be 64 bits, which is more than enough to avoid address collisions even when the addresses would be used non-sparingly, and the autoconfiguration mechanism contains a duplicate address detection protocol for avoiding address collisions. However, the Ethernet MAC address size is only 48 bits, and it consists of 24 bit Organizational Unique Identifier (OUI) and 24 bit value for generating the unique address for the particular vendor organization. 802.11 based WLAN also use the same MAC address format. Thus, frequently used random addresses could create collisions in very large local networks, if we assume that we leave the OUI part unmodified.

The collision probability has been previously [18] analyzed for 27 bit addresses. For example, if there are 1000 clients in the network, it follows from the birthday paradox that the probability of collision is 0.65 in 24 hour intervals if the clients change their MAC address every five minutes. The authors of the analysis also proposed a mitigation mechanism by reverse ARP. We leave implementation and analysis the applicability of MAC address collision mitigation mechanisms for further work.

Although our tests showed that with some applications the address usage can be considerably large, the addresses can be aggregated with the introspection feature. For example, for providing unlinkability, it might not make a difference if all Skype traffic seem to originate from the same host because the traffic is encrypted end-to-end.

**Dynamic DNS** Dynamic DNS can be used to covertly track users [19]. When a host changes location, it updates the new IP address to the dynamic DNS server. Thus, peers knowing only the DNS name can contact the host. Researchers have implemented a software that use these IP addresses to track users with a IP geolocation service [19].

Our approach does not prevent from tracking and identifying the user if the host uses dynamic DNS as described. However, our approach reduces the information available to a local attacker. The user can remain reachable and identifiable, but other services and applications that do not rely on the dynamic DNS cannot be linked to the user or the host.

**Operating system and physical device fingerprinting** An attacker could try to identify the device by operating system fingerprinting with a tool such as nmap [30]. These techniques can be mitigated with host-based protocol scrubbing [41] or normalization [20]. These techniques are already implemented for operating systems such as FreeBSD and Linux, and we consider integrating a protocol scrubber to our system to be straightforward.

Also, the physical and link layers could still link the applications together, for example, with analog signal fingerprinting [14] or by the device driver behavior with e.g. WLAN [12]. However, mitigation of these techniques is beyond the scope of this work, and already the authors of WLAN fingerprinting proposed techniques for preventing it [12]. Also, it might be possible to attack the system we have implemented with a new low-level traffic analysis technique directly designed for revealing the virtualization. The current implementation does not do anything for the Ethernet or WLAN interface, thus, it uses the normal collision avoidance techniques that have been specified in the standards and implemented by the device manufacturer. Thus, with time, and capturing enough of the traffic, an attacker could observe which ones of the MAC addresses are not colliding and statistically deduce the address spaces that belong to the same physical device.

**Out-of-band information** If the attacker has out-of-band information, the approach might not provide much privacy. If a user walks alone to an Internet cafe, and is the only customer present, the owner of the cafe knows that any new traffic originates from the user and especially if authentication to the network is required. However, contrary to existing approaches, we do not need to rely on the actions of other users in the network, as is the case in anonymity systems [9], where attracting other users to use the network is exceedingly important.

**Application layer traffic and duplicate identifiers in flows** The final limitation of the presented approach is that although we protect the linking of identifiers and some application data, there are examples of information flow that the approach does not help with. Consider an example where the user with set of identifiers $A$ downloads a document from an FTP site. The user then emails the document with set of identifiers $A'$ with a Web based email client. Now, it is possible to link the identifier sets $A$ and $A'$ if there is only a little time between these two transactions, because exactly the same file is sent and received by both $A$ and $A'$.

We sketch a possible approach to the above problem. The application traffic could be recorded by hashing the data and adding it to a database. If similar traffic/document is passed to another set of identifiers, the user could be notified. Naturally, this increases significantly the complexity of the system, and conflicts with the original goals of user-transparency. We leave this problem for further work.

### 7.2 Further remarks

We now contrast our approach to a a distributed infrastructure to provide sender and receiver anonymity. In the related work, we already have contrasted our approach to infrastructureless approaches. We limit our discussion to Tor [10], since it can be considered as the de facto way to achieve sender and receiver anonymity in the Internet and thus protect the identity of the user. Although Tor is relatively simple to install, it hinders the user experience with increasing the latency of e.g. Web browsing. Further, Tor has an alarming usability problem, because for some reason, many users tend to believe that Tor provides end-to-end encryption and do not protect their traffic accordingly [43]. Even though we acknowledge that Tor is probably the best current deployed way to protect your identity in the network, our work explores a solution space where you do not trust the network at all. Tor relies on distributed infrastructure, but we desired to find out how much privacy users can receive without resorting to any kind of infrastructure. In fact, our approach can be seen as complementary to Tor as discussed in the related work.

It is, however, important to distinguish who we are trying to protect from. If we can assume that the access network provider can be trusted, and the privacy protection mechanisms need to be in place for attackers one hop or further away from the access network much simpler mechanisms suffice. Traditional NATs, a new IP address for every new TCP or UDP flow [6] can provide enough privacy against most attackers residing outside the access network.

Our tests showed that the approach does not hinder the user experience in any noticeable way, provided that the network allows the use of multiple addresses. We also required changes only to a single operating system and did not assume any infrastructure. The users can also remain reachable even though the protocol stack virtualization is in place. For example, for a given set of identifiers, the system can enable dynamic DNS or SIP registration for VoIP calls to remain reachable.

As final note, we think that this approach should be interesting also because it does not seem to be dual use technology. For example, Tor can be used for illegal purposes easily in addition to protecting the privacy of ordinary users, but our approach simply provides unlinkability for different traffic sent to the network by the user's device. This means that if the user does something illegal with one set of identifiers, law enforcement can at least pinpoint the network where the traffic originated.

## 8 Further Work

We presented limitations and outlined possible approaches for solving them in the discussion section. In summary, our implemented system should be integrated with protocol scrubbers and especially MAC and application layer attacks are a concern. What should be the lifetime of the pseudorandom addresses used with the protocol stack virtualization is an open problem. Clearly, for example, when the user changes an access point in a network where MAC address based authentication is not used, at least the MAC layer identifiers should be changed. If also the IPv6 network prefix changes, the IP addresses should naturally also be changed.

We plan also to integrate the architecture presented in this paper to (secure [5]) network location awareness (NLA). We have previously proposed [4] that NLA can be used to mitigate many service discovery related information leaks. One option for further work is to implement the system on Windows in addition to the current Linux implementation. Windows Filtering Platform and its application-based packet filtering should provide further insight for both NLA and protocol stack virtualization as means to protect the privacy of the mobile user.

## 9 Conclusions

We have presented an approach for identity privacy protection that could significantly change how applications use the network addresses of a networked operating system, even though the changes to the operating system are almost negligible as we have shown with the user space implementation. One of the main benefits of the approach is that when properly implemented, applications do not need changes and the user experience does not change. The users can still lose their privacy with some applications, but it does not help a third party to deduce anything else about the user. To the best of our knowledge, this is the first approach that attempts to provide privacy without infrastructure for the whole mobile networked system by making changes to only a single host and not to both peers.

## Acknowledgments

Janne Lindqvist's thinking on the topic was much evolved by sharing the work space for six months during 2007 with Teemu Koponen at the International Computer Science Institute, Berkeley, CA. Teemu's suggestions and comments have significantly helped to focus this work. The authors are also grateful for Tuomas Aura for a rewrite of an earlier version of the introduction section and for providing thorough comments on how to write proper introductions. We also thank George Danezis and Michael Roe for pointing out some of the limitations, and also the anonymous reviewers and our shepherd XiaoFeng Wang for their comments.

During this work, Janne Lindqvist was partially funded by scholarships from Nokia Foundation, and Research and Training Foundation of TeliaSonera Finland Oyj.

## 10 References

[1] M. Abadi and C. Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, Sept. 2004.

[2] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)*, 7, May 2004.

[3] J. Arkko, P. Nikander, and M. Näslund. Enhancing Privacy with Shared Pseudo Random Sequences (preliminary version). In *Security Protocols, 13rd International Workshop*, Apr. 2005.

[4] T. Aura, J. Lindqvist, M. Roe, and A. Mohammed. Chattering laptops. In *8th Privacy Enhancing Technologies Symposium (PETS)*, July 2008.

[5] T. Aura, M. Roe, and S. J. Murdoch. Securing Network Location Awareness with Authenticated DHCP. In *3rd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Sept. 2007.

[6] T. Aura and A. Zugenmaier. Privacy, Control and Internet Mobility. In *Security Protocols, 12th International Workshop*, Apr. 2004.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP*, Oct. 2003.

[8] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, Feb. 1981.

[9] R. Dingledine and N. Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *Workshop on the Economics of Information Security*, June 2006.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*, Aug. 2004.

[11] N. Droux, S. Tripathi, and K. Belgaied. Crossbow: Network virtualization and resource control. http://www.usenix.org/events/usenix07/posters/droux.pdf.

[12] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *15th USENIX Security Symposium*, July/August 2006.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP*, Oct. 2003.

[14] R. Gerdes, T. Daniels, M. Mina, and S. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *Network and Distributed System Security Symposium (NDSS)*, Feb. 2006.

[15] P. M. Gleitz and S. M. Bellovin. Transient addressing for related processes: improved firewalling by using IPV6 and multiple addresses per host. In *10th USENIX Security Symposium*, Aug. 2001.

[16] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Research in Security and Privacy*, Apr. 1982.

[17] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In *Workshop on Information Hiding*, May/June 1996.

[18] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: a quantitative analysis. *Mob. Netw. Appl.*, 10(3):315–325, 2005.

[19] S. Guha and P. Francis. Identity Trail: Covert Surveillance Using DNS. In *Workshop on Privacy Enhancing Technologies (PET)*, June 2007.

[20] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *10th USENIX Security Symposium*, Aug. 2001.

[21] S. Ioannidis, S. Sidiroglou, and A. D. Keromytis. Privacy as an Operating System Service. In *1st USENIX Workshop on Hot Topics in Security (HotSec)*, July 2006.

[22] T. Jiang, H. J. Wang, and Y.-C. Hu. Location privacy in wireless networks. In *MobiSys*, June 2007.

[23] S. Katti, J. Cohen, and D. Katabi. Information Slicing: Anonymity Using Unreliable Overlays. In *NSDI*, Apr. 2007.

[24] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2), April/June 2005.

[25] J. Lindqvist and L. Takkinen. Privacy management for secure mobility. In *Workshop on Privacy in the Electronic Society (WPES)*, Oct. 2006.

[26] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *USENIX Annual Technical Conference*, May/June 2006.

[27] S. J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *CCS*, October/November 2006.

[28] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, 2000.

[29] T. Narten, R. Draves, and S. Krishnan. RFC 4941: Privacy Extensions for Stateless Address Autoconfiguration in IPv6, Sept. 2007. Status: Draft Standard.

[30] Nmap. http://www.insecure.org/nmap/.

[31] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *MobiCom'07*, Sept. 2007.

[32] J. Peterson. RFC 3323: A Privacy Mechanism for the Session Initiation Protocol (SIP), Nov. 2002.

[33] Planetlab. https://www.planet-lab.org/.

[34] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, Nov. 1998.

[35] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol, June 2002.

[36] U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In *CCS*, October/November 2006.

[37] P. Srisuresh and K. Egevang. RFC 3022: Traditional IP Network Address Translator (Traditional NAT), Jan. 2001. Status: Informational.

[38] G. Su and J. Nieh. Mobile communication with virtual network address translation. CUCS-003-02, Columbia University Department of Computer Science, Feb. 2002.

[39] S. Thomson, T. Narten, and T. Jinmei. RFC 4862: IPv6 Stateless Address Autoconfiguration, Sept. 2007. Status: Draft Standard.

[40] VMware. http://www.vmware.com.

[41] D. Watson, M. Smart, G. R. Malan, and F. Jahanian. Protocol scrubbing: Network security through transparent flow modification. *IEEE/ACM Transactions on Networking*, 12(2), Apr. 2004.

[42] A. R. Yumerefendi, B. Mickle, and L. P. Cox. TightLip: Keeping Applications from Spilling the Beans. In *NSDI*, Apr. 2007.

[43] K. Zetter. Rogue nodes turn tor anonymizer into eavesdropper's paradise. *Wired*, Sept. 2007.