Adaptive Protocol Switching Using Dynamically Insertable Bumps in the Stack

Devin J. Pohly
SIIS Laboratory
Pennsylvania State University
University Park, PA 16802
Email: djpohly@cse.psu.edu

Charles Sestito
SIIS Laboratory
Pennsylvania State University
University Park, PA 16802
Email: sestito@psu.edu

Patrick McDaniel
SIIS Laboratory
Pennsylvania State University
University Park, PA 16802
Email: mcdaniel@cse.psu.edu

***Abstract*—Mobile networked devices face a unique set of challenges, particularly when used in an adversarial environment. These devices must be able to respond to changing circumstances in both their physical and network environment. We present DIBS, a system that enables devices to dynamically insert bump-in-the-stack protocols in response to changing conditions. We evaluate this system by measuring connection throughput and CPU usage as it switches between an unmodified protocol stack, a VPN, and a high-secrecy multichannel stack. Our experiments show that DIBS is able to switch seamlessly between protocols without interrupting ongoing connections or introducing additional network or processing overhead. This ability will enable devices to adapt communications rapidly in response to changing cybersecurity and physical environments.**

***Index Terms*—Agility, network protocols, mobile devices, communications.**

I. Introduction

Mobile networked devices face a unique set of challenges, particularly when used in an adversarial environment. A few examples of concerns when developing mobile technology include security, battery usage, and reliability. These concerns may conflict with each other, such as additional security increasing battery and CPU usage, requiring intelligent trade-off decisions based on the current environment. Devices must be able to quickly take measures to remain secure in the face of changing environmental factors and varying threats. Countermeasures to these threats must intelligently balance the needs of the user, and the device should be able to do all of this without interrupting any ongoing transmissions.

The ability to adapt is crucial to any sort of mission-critical communication because different scenarios can require completely different functionality. The need for adaptability is especially prevalent during military action. When on the ground, soldiers must be able to communicate effectively with each other and with their base station. Security measures must be implemented to prevent the enemy from intercepting critical information about the soldiers and their mission. Mobile devices have a limited supply of battery power, so it must be conserved when possible. When communications are less sensitive, devices can optimize to save battery or prioritize other concerns. Consider a mobile device in active use on the ground, where a number of scenarios may happen. First, clouds may accumulate overhead and block satellite communication, forcing the device to adapt to communicate by other means. The device may be running low on battery, requiring it to conserve power in order to extend communication time with auxiliaries. There are also circumstances, such as checking a weather forecast, which have no special security or performance requirements and would be best done with minimal power consumption. Without the ability to quickly adapt to changing conditions, the soldiers who use these devices can become more susceptible to lost transmissions, dead batteries, or compromised communications.

In this paper, we present Dynamically Insertable Bumps in the Stack (DIBS), a system which enables protocol-switching maneuvers in response to changing physical or cyber conditions. These maneuvers take place without interrupting ongoing communication, and the system can be deployed entirely without modifications to existing applications, IP networks, or system kernels. DIBS gives devices the ability to switch protocols to adapt to changing circumstances. Needs such as security, power, and reliability can be dynamically balanced to aid soldiers in successfully carrying out their mission.

One frequent approach to solving these problems is creating protocols that address particular concerns [1]–[3]. In order to adapt to changes in the environment, we need to implement a technology that addresses these concerns dynamically. We show that DIBS achieves this, and that it does so without any additional overhead. In our experiments, DIBS switches fluidly between an unmodified protocol stack, a secure multichannel protocol, and an SSL VPN without losing a connection. Each of these bump-in-the-stack protocols provides different properties, and with DIBS the device can choose adaptively to address the various challenges of mobile networked devices.

II. Architecture

DIBS is a system for creating and manipulating dynamically insertable bump-in-the-stack protocols (or DIBs) in the network stack of a running device, allowing it to change protocols in response to changing conditions. A high-level overview of the system's functionality is given in Figure 1: each host is running one or more protocols as DIBs, and the system can select the protocol over which any given connection is routed, as well as change this selection over time. The entire maneuver happens transparently to communicating applications, with no interruption to the connection between hosts.

978-1-5090-0073-9/15/$31.00 ©2015 IEEE

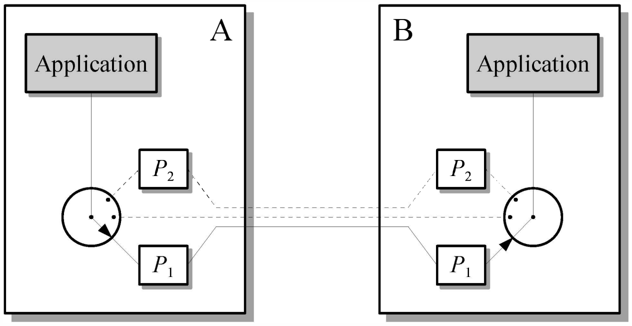

Fig. 1. Overview of DIBS functionality, shown with bump-in-the-stack protocol P_1 selected for a given connection between hosts A and B

A. Required Properties

In order to make DIBS adaptable to as many different circumstances as possible, we require that it be designed to be highly transparent, flexible, and agile. Transparency will ease deployment and allow the system to be used with many different applications. Flexibility will allow it to address a variety of different situations, and agility will allow it to adapt as the situation changes.

The first required property is transparency. DIBS should require no modification to existing applications, no customizations to the kernel, and no additional network hardware. Applications should be oblivious to the fact that different protocol stacks are being used. Furthermore, the system should not introduce significant network or processing overhead beyond what the protocols themselves would require. This property ensures that deployment is non-invasive and requires minimal effort. It also allows DIBS to be layered with other protocols and applied to traffic from any network application, regardless of whether its source code is available for modification.

Note that we are referring here to transparency of implementation, i.e., that applications require no modification to benefit from DIBS. The effects of changing protocols on network traffic throughput and timing are of course visible to the communicating programs (as we illustrate in the experiments in Section III). However, the visible effects are essentially the same as those of normal in-network fluctuations such as congestion or route changes. Essentially, since applications are already designed to handle these expected network conditions, even the aspects of DIBS which are visible to applications will not require any modification to their implementation.

The second property that DIBS must fulfill is flexibility, both in selecting traffic and in selecting the protocol over which to send it. The system should be able to select individual connections based not only on the destination host address, but on a variety of other criteria including network port, user, and time of day. These selection criteria should be modifiable, as they will change over time in response to changing threats. DIBS should also support arbitrary, custom bump-in-the-stack protocols, so that new, experimental, or classified protocols can be implemented easily and without relying on kernel support.

One essential aspect of this flexibility is that the modifications to the configuration are not dictated by DIBS itself. Protocol maneuvers can be executed by any user or process on the device which is granted network administration capabilities (typically only the root user). This lends itself to diverse possibilities. For example, a specially privileged application, or even a hardware switch, could be manually operated by the device user to change protocols. Maneuvers could be effected by an administrative process which receives instructions from central command. A protocol switch could even be prompted directly by alerts from an intrusion detection system or environmental sensor which indicates a change in the cyber or physical environment. DIBS is designed specifically to avoid limiting these possibilities.

Finally, DIBS must provide enough agility to change protocols for any traffic at any point in time. It must be able to do this without interrupting ongoing network connections, since doing so would break transparency to applications. However, it is not acceptable to wait for connections to complete before applying protocol changes, as this would delay the response to a new threat, and any long-running connections would not be affected. Therefore, the system must be able to reroute traffic to other protocols immediately and without interruption.

All of these properties are achieved by constructing DIBS as a novel organization of a limited set of building blocks: only those networking features which are already present and enabled in the default Linux routing infrastructure. The following sections describe how these features are used in a non-traditional fashion to achieve the system goals. DIBS uses these mechanisms first to adapt the protocols themselves, then to select and intercept traffic, and finally to redirect it transparently between different protocol implementations, all while modifying nothing other than routing and firewall configuration. In particular, this approach obviates the need for a separate meta-protocol to manage bumps, and it also avoids the overhead which could come from introducing new components into the network fast path.

B. Adapting Protocols to DIBS

The first step in adding a bump is implementing the protocol in such a way that it can be used by DIBS. Adding protocol-implementing middleboxes (e.g., VPN gateways) to the network would be a simple solution to this problem, were it not for the requirement that DIBS be deployable without modifying existing networks. Likewise, the bump could easily be added to an unmodified kernel using loadable kernel modules, but this could not be selectively applied to applications unless they are modified to request it.

In order to satisfy its design goals, therefore, DIBS takes a different approach, in which each protocol bump is handled by a userspace program rather than in hardware or the kernel. Userspace implementation simplifies the development of new protocols by allowing them to be tested without the possibility of a bug crashing the entire system. It has been used successfully for experiments and prototyping with protocols such as SCTP [4] and AODV [5]. Some bump-in-the-stack protocols even use this approach for their primary

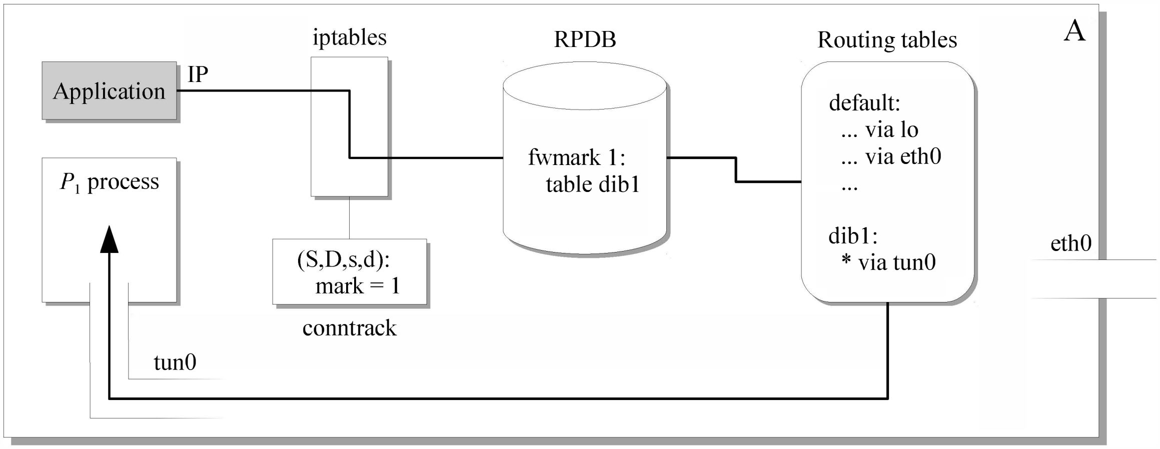

Fig. 2. Using policy routing to intercept a specific connection on host A and route it via DIB protocol P_1

implementations, due to its ease of testing and maintenance. Two examples of this (both of which are used in the evaluation of DIBS) are the TLS-based OpenVPN protocol [6] and the multichannel MICSS protocol [7]. This benefits DIBS in that new and experimental bumps can be developed easily, as the only requirement is a userspace protocol implementation.

Writing a protocol in userspace can be accomplished simply on Linux by making use of the kernel's TUN/TAP driver [8], which allows a process to create or attach to a virtual network interface. The process will receive and handle any packets which are handed down from higher layers via this interface, and it can then manipulate or encapsulate them according to the new protocol. When data arrives over the protocol, the process can then use the interface to inject higher-layer data back into the regular IP stack.

C. Intercepting Traffic

Once there is a custom protocol process awaiting packets on a virtual interface, DIBS must select and route traffic via this interface so that it can be handled by the protocol implementation. Typical network bumps such as VPNs do this simply by adding an entry to the system routing table. However, the Linux routing tables only match packets by their destination host address or prefix. This is far too coarse to meet the flexibility requirements of DIBS. In particular, it cannot even be used to select one specific connection; all traffic destined for a particular host is routed via the same interface. Considering that the custom protocol will likely generate traffic to send to the same destination as what was intercepted, attempting to use routing tables alone will often cause a loop in which the output of the protocol is routed back to it as input.

The problem is that, in a typical routing configuration, outbound network interfaces are chosen *exclusively* by routing table entries, so the system cannot direct traffic to the virtual interface without using the routing table in some way. For DIBS to provide the fine-grained traffic selection it needs while still ending up at the routing table, it must coordinate several other features of the Linux network stack. Some of these, like iptables, are very common; others, such as the Routing Policy Database, are infrequently used outside of advanced routing environments.

IP packets from applications begin their traversal of the Linux network stack at the iptables routing and filtering framework. Rules in iptables can be used to identify traffic very selectively—not only by source and destination address and port, but by any other field in packet headers, as well as based on external properties such as the originating user or the current time. This easily satisfies the requirement for flexible traffic selection.

With iptables and routing tables, DIBS has a means of selecting traffic to be intercepted and a means of sending traffic to the virtual interface, but it still needs a means of connecting the two. To accomplish this, it exploits three of the Linux kernel's policy routing features: packet marking, the Routing Policy Database (RPDB), and multiple routing tables. As described above, a single routing table entry in the default table is not sufficient to separate traffic which should and should not be routed to a custom protocol. However, if each bump is given a separate, non-default routing table which routes *all* traffic to the corresponding virtual interface, then a more general routing policy can be used to decide which bump will be used.

The first step is to set up these separate routing tables. For each protocol P_i which is to be handled by DIBS, a table called dib*i* is created and populated with a single rule directing all traffic to the protocol's TUN/TAP interface. Then a rule is entered in the RPDB to select that table for any packets marked by iptables with the corresponding number. For example, routing for two custom protocols is set up with the following commands:

```
ip route add table 1 default via tun0
ip rule add fwmark 1 table 1
ip route add table 2 default via tun1
ip rule add fwmark 2 table 2
```

The final step is the configuration of iptables to set the firewall mark ("fwmark") on packets according to the protocol they should use. For each class of traffic which should be routed via DIBS, an output rule is added to the iptables firewall to select and mark it. For example, this configuration:

```
-p tcp --dport 80 -j MARK --set-mark 1
-m owner --uid-owner tom -j MARK --set-mark 2
```

would route all HTTP traffic over protocol P_1 and all traffic from the user “tom” over protocol P_2.

Figure 2 shows the interaction between all of the routing features required to make DIBS work. An IP packet is generated by the network application and enters the Linux protocol stack. The iptables firewall marks the packet with the number of the appropriate DIB, if any, based on the given selection criteria (and potentially the connection tracking mechanism detailed in the next section). This mark is used by the Routing Policy Database to choose an alternate routing table, which then sends all traffic to the virtual TUN/TAP interface. At the other end of this interface is the DIB protocol implementation, which then processes, encapsulates, and transmits its own traffic as the DIB protocol dictates.

D. Switching Protocols

Although transparency and flexibility are certainly important, the most significant advantage of the DIBS architecture is the ability to enable, disable, or switch between protocol bumps on the fly. Given the policy routing infrastructure set up in the previous sections, the only change that needs to be made in order to change the DIBS configuration for a particular connection or class of traffic is the fwmark being assigned to its packets. This means a bump in the stack can be added, removed, or changed for any given traffic with a *single iptables command*. While this is impressive in itself, the transparency of DIBS adds a further benefit. Since the changes happen at the network layer and are transparent to applications, protocol-switching maneuvers can be carried out with no interruption whatsoever to ongoing connections.

This configuration allows each host to determine when outgoing traffic should be moved to a different protocol, but the decision for return traffic must be made on the other host. Ideally, if one host actively switches to protocol P_1, the other should follow suit. This behavior can also be achieved with iptables by taking advantage of its connection tracking module. This feature stores a small amount of state for each connection made by the host, including a freely usable “ctmark” akin to the fwmark for packets, but which is associated with the entire connection rather than individual packets. Rather than setting the fwmark for every matching packet, DIBS can instead set the ctmark once and let iptables copy it to each packet belonging to the connection.

A clever application of this connection marking feature allows DIBS to switch protocols automatically in response to a switch on the remote host. As soon as traffic begins to arrive from the remote host on a new virtual interface, DIBS updates the ctmark for the corresponding connection so that any return traffic is routed back through the same protocols. This can be done very succinctly in iptables by creating a single input rule for each protocol which selects packets based on the interface on which they arrived and sets the connection mark accordingly:

```
-i tun0 -j CONNMARK --set-mark 1
-i tun1 -j CONNMARK --set-mark 2
```

In addition, the connection mark can be manipulated for individual connections without having to add a full-fledged rule to the iptables firewall ruleset. One utility for doing this is the “conntrack” command-line utility available as part of the standard conntrack-tools package from the iptables development project. Given a utility such as this, applications can reroute connections individually based on properties which are not even available to iptables, such as the sensitivity of data being transmitted, or simply by user request.

III. Evaluation

For our experimental setup, we deployed DIBS on a pair of hosts connected to a quiescent network. Each host was a Dell Precision T7600 workstation running Arch Linux with kernel 3.19.3 and configured with two DIB protocols in addition to the default unmodified network stack. The first protocol was a secure secret-sharing multichannel system based on MICSS [7]. The second DIB was OpenVPN’s TLS-based virtual private networking protocol. Since OpenVPN already uses TUN/TAP interfaces with a userspace process to implement its protocol, it was a natural candidate for adaptation as a DIB. To support these particular DIBs, the hosts were connected by three independent gigabit Ethernet links. Unmodified and OpenVPN communications used only the first link, whereas the multichannel protocol used all three simultaneously.

The experiment was carried out as follows. Both DIB processes were started and allowed to run in the background, and the system was initially configured to send all communication over the unmodified protocol stack. The Netperf benchmarking tool [9] was used to generate a stream of TCP traffic from one host to the other. Over the course of the 30-second experiment, the DIBS configuration was modified once every ten seconds. At time 0, the system was in its initial state (unmodified network stack). At time 10, the iptables rule was changed to use the secure multichannel protocol instead, and at time 20 it was again changed to route the Netperf traffic over the VPN instead.

We collected two datasets during each run of this experiment. The first dataset was a packet trace from each of the DIB virtual interfaces as well as the physical interface (for unmodified traffic). These packet traces were merged together, and the sequence numbers in the TCP packets were used to calculate the average connection throughput at 0.25-second intervals. The second dataset was CPU usage information calculated by reading the Linux `/proc/stat` interface at 0.5-second intervals.

The results of the experiment are shown in Figure 3, with the CPU usage superimposed on the network throughput. Each time the DIBS configuration is changed, the difference in protocol properties appears prominently. To start, the unmodified protocol stack provides the highest performance and

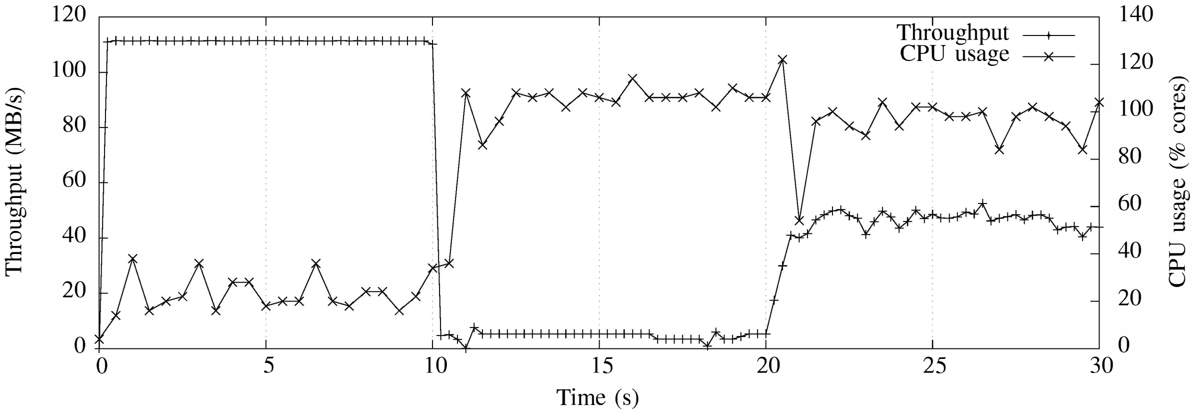

Fig. 3. Connection throughput and CPU usage when switching between unmodified communication (0–10), multichannel (10–20), and VPN (20–30)

TABLE I
BASELINE PERFORMANCE OF INDIVIDUAL DIB PROTOCOLS

DIB	Throughput (MB/s)	CPU usage (%)
DIBS disabled	117	10
Regular TCP/IP	117	10
Multichannel	6	108
VPN	50	91

lowest CPU usage, but adds no security. This would be ideal for preserving battery life during activities which are not sensitive. The switch to the multichannel DIB demonstrates the other end of the spectrum, sacrificing both throughput and processing for maximum information-theoretical security. Multichannel security could be used for mission-critical communications which require security as a top priority. Changing from multichannel to the VPN DIB shows an increase in throughput and a small drop in processor usage, indicating that it could be use when a balance between these two extremes is desired.

In order to demonstrate that there was no additional overhead incurred by the use of DIBS, we also measured the throughput and CPU usage of each protocol in isolation, as well as of the system without DIBS interception. This baseline is shown in Table I. The values for throughput and CPU usage without DIBS match closely to the graph of the previous experiment, and the results for disabling DIBS and running DIBS with no added protocol were indistinguishable. Since DIBS is constructed from routing mechanisms which are already used in normal system operation, this result is as expected.

IV. Related Work

Current techniques to solve the challenges of the use of mobile networked devices in adversarial environments tend to focus on a single challenge such as security or reliability [1]–[3], [10]. These protocols are beneficial in specific scenarios, but the ability to switch between these precisely engineered protocols based on a situational change is needed to protect the security, efficiency, and effectiveness of communications.

A number of mobile protocols have been proposed in the past decade. Simple Relay Enabled MAC (SRMAC) was created to counteract the effects of signal power attenuation with distance to increase the throughput of transmissions [2]. However, SRMAC requires multiple transmissions of a packet, which delays the overall transmission time. Similarly, Link-16K was proposed to reduce the effect of long propagation delays and the overhead of acknowledgments for large transmission such as imagery, but it is not optimized for all data transmissions [1]. Public Key Dynamic Signcrypted Identification Protocol (PK-DS-ID) was developed to reduce CPU usage with secure communications [3], although the use of PK-DS-ID requires additional hardware such as a smart card in order to work. MANET Anonymous Peer-to-peer Communication Protocol (MAPCP) increases anonymity of peer-to-peer connections, but requires overhead and delays transmission [10]. While these protocols and designs, along with an array of other communication protocols, are able to solve a few problems, it would be optimal to use different protocols to cater to different situations.

Other research has focused on protocol-switching capabilities. Hardware was shown to be able to switch between protocols, but required unwanted overhead costs [11]. Meta-protocols have been proposed to combine properties of existing protocols [12], yet this work did not implement a system to take advantage of its findings. A survey of cyber moving target defenses was able to identify strengths and weaknesses in several dynamic network protocols to increase security of communications while requiring additional overhead or compromising other resources [13]. Although dynamic protocol switching has been proposed, there have not been efforts to allow a device to be customized to dynamically switch between protocols based on individual specifications.

V. Conclusion

The unique challenges faced by mobile networked devices in adversarial environments require the ability to cater to the needs of a variety of situations. Devices needed to be able to respond to the changing physical and network environments around them. In this paper, we have presented DIBS, a system that enables devices to dynamically insert bump-in-the-stack protocols in response to changing conditions. We were able to evaluate the performance of the system by measuring the connection throughput and CPU usage as connections are moved between an unmodified protocol stack, a VPN, and a high-secrecy multichannel stack. The experiments have yielded that DIBS is able to seamlessly switch between network protocols without affecting ongoing transmission or adding network or processing overhead. DIBS has made it possible for devices to dynamically add, remove, or change bumps in their own network stack with no additional overhead, enabling a dynamic change of protocols in reaction to specified inputs. DIBS is the solution needed to adapt communications rapidly in response to changing cybersecurity and physical environments.

References

[1] H. Baek, S. Ko, J. Lim, and I. Oh, "Enhanced mac protocol with new packing for imagery transmission in link-16," in *Military Communications Conference (MILCOM), 2014 IEEE*, Oct 2014, pp. 891–896.

[2] S. Kim and W. Stark, "Simple relay enabled mac (srmac) protocol for cooperative communication," in *Military Communications Conference, MILCOM 2013 - 2013 IEEE*, Nov 2013, pp. 175–180.

[3] H. Elkamchouchi, A.-A. Emarah, and E. Hagras, "A new public key dynamic signcrypted identification (pk-ds-id) protocol using smart cards," in *Radio Science Conference, 2007. NRSC 2007. National*, March 2007, pp. 1–10.

[4] B. Penoff, A. Wagner, M. Tuxen, and I. Rungeler, "Portable and performant userspace SCTP stack," in *Computer Communications and*

[4] B. Penoff, A. Wagner, M. Tuxen, and I. Rungeler, "Portable and performant userspace SCTP stack," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE, 2012, pp. 1–9.

[5] E. M. Royer and C. E. Perkins, "An implementation study of the AODV routing protocol," in *Wireless Communications and Networking Confernce, 2000. WCNC. 2000 IEEE*, vol. 3. IEEE, 2000, pp. 1003–1008.

[6] B. Hoekstra, D. Musulin, and J. J. Keijser, "Comparing TCP performance of tunneled and non-tunneled traffic using OpenVPN," *Universiteit van Amsterdam System & Network Engineering, IEEE*, 2011.

[7] D. J. Pohly and P. McDaniel, "MICSS: A realistic multichannel secrecy protocol," Institute for Networking and Security Research, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, Tech. Rep. NAS-TR-0179-2014, Oct. 2014.

[8] M. Krasnyansky, M. Yevmenkin, and F. Thiel, "Universal TUN/TAP device driver," https://www.kernel.org/doc/Documentation/networking/tuntap.txt, 2002.

[9] R. Jones, "The Netperf homepage," http://www.netperf.org/.

[10] C.-C. Chou, D. Wei, C.-C. Kuo, and K. Naik, "An efficient anonymous communication protocol for peer-to-peer applications over mobile ad-hoc networks," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 1, pp. 192–203, Jan 2007.

[11] G. Carvajal and S. Fischmeister, "A tdma ethernet switch for dynamic real-time communication," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, May 2010, pp. 119–126.

[12] X. Liu, R. Van Renesse, M. Bickford, C. Kreitz, and R. Constable, "Protocol switching: Exploiting meta-properties," in *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society, 2001, pp. 0037–0037.

[13] H. Okhravi, M. Rabe, T. Mayberry, W. Leonard, T. Hobson, D. Bigelow, and W. Streilein, "Survey of cyber moving target techniques," DTIC Document, Tech. Rep., 2013.