Securing the Insecure Link of Internet-of-Things Using Next-Generation Smart Gateways

Syed Rafiul Hussain*, Elisa Bertino*, Shahriar Nirjon†
*Purdue University, †University of North Carolina Chapel Hill
Email: *hussain1@purdue.edu, *bertino@purdue.edu, †nirjon@cs.unc.edu

***Abstract*—Since low-cost IoT devices have limited computing resources and are often unable to guarantee sufficient security, it is imperative to ensure secure communication between IoT gateways and cloud. In this paper, we propose a flow-level adaptive mobile VPN solution specifically tailored for IoT ecosystems, called *AdamVPN*, which adapts its configuration dynamically at runtime in order to improve IoT gateway's application-level throughput while conforming to the security and privacy guarantees simultaneously. Our deployment experiments in both Wi-Fi and cellular environments demonstrate that *AdamVPN* significantly improves throughput by 2.75x–3.0x for Wi-Fi and 1.8x–2.16x for cellular networks when compared to OpenVPN.**

***Index Terms*—IoT, Gateway, VPN, Security, Privacy.**

I. Introduction

In IoT ecosystems, private data collected by end devices are sent to the cloud for storage and analysis. The end devices, such as smart lights, smart locks, drones, or mobile robots, used in home, agriculture, battlefields, and industry automation, however, often do not have the full network stack required for Internet connectivity and/or the processing capability and the battery power for performing expensive cryptographic operations [1], [2]. A gateway device acting as a proxy addresses this problem by aggregating and transmitting end devices' data to and from the cloud over TCP/IP. Ensuring secure communication between an IoT gateway and the cloud is, therefore, critical to the overall security of the IoT ecosystem and is thus the focus of this paper.

Though traditional SSL/TLS based solutions (e.g., HTTPS and VPN) seem to be effective for securing such communication link, they have been shown not resilient [3] against sophisticated privacy attacks which allow attackers to infer privacy-sensitive in-home activities by analyzing Internet traffic from smart homes. Also, such SSL/TLS based solutions are often difficult to employ for IoT ecosystems due to the following limitations: (1) A single gateway application can barely comply with different encryption/authentication policies required by different types (e.g., lights vs. cameras) and families (Phillips vs. Samsung) of end devices. Therefore, use of a single statically defined tunnel configuration, i.e., fixed encryption and authentication schemes for all classes of devices, applications, and network context does not allow one to optimize the network and computing resources [4]. (2) A dedicated VPN tunnel for each IoT application and traffic type (e.g., audio, video, and text data for Virtual Reality) induces significant processing latency, bandwidth and memory overheads, and network congestion. (3) 'Hard-reset' is required every time there is a change in the configuration. (4) Use of no encryption or weak cryptographic constructs [5], [6] for faster data transfer. Given all these factors combined, VPN or SSL/TLS clients on an IoT gateway would yield an impractical network throughput with weak privacy and security guarantees. In the light of above discussion, this paper addresses the following research question: Is it possible to develop a VPN solution that is capable of seamlessly changing the configuration at runtime to deploy customized tunnel configurations to achieve higher bandwidth and strong security and privacy guarantees for a diverse set of IoT applications?

To address the above research question, we propose a flow-level adaptive mobile VPN, called *AdamVPN*, which is capable of adapting its configuration dynamically at runtime based on the type of data traffic (e.g., text, real-time voice and video, compressed/uncompressed, ciphertext/plaintext), the current network context (e.g., client's network interface and throughput), and the given security policies (e.g., AES-128-CBC for encryption/decryption and SHA128 for signature generation/verification) of the data and IoT devices. We explore the parameter space of VPN configurations to identify *security-sensitive* as well as *throughput-sensitive* parameters. We assume a *base security configuration* that ensures a level of security that is presumably unbreakable in the foreseeable future, and an *application default configuration* that represents the security policy (e.g., encryption and signature algorithm, key size, and throughput related parameters) recommended by the IoT application.Based on these, we argue that the strength of security sensitive parameters used by *AdamVPN* is always the maximum parameter strength between the base security setting and the application default. Hence, even if an IoT application does not require the data to be encrypted over a VPN tunnel, *AdamVPN* applies its base security configuration and thus protects data confidentiality by applying encryption.

AdamVPN leverages an SSL-based open source VPN solution, and implements two novel algorithms: (1) a *policy* to select a suitable configuration, and (2) a *mechanism* to dynamically adapt to the configuration selected based on the policy. *AdamVPN* intercepts IP packets from the SSL-based VPN and by combining information about the packet type (obtained using a custom traffic classifier) adapts to a suitable VPN configuration according to the chosen mode of operation. We perform an in-depth evaluation to quantify the overheads of *AdamVPN*, and to empirically measure the effect of traffic types and network characteristics on its performance.

We deploy *AdamVPN* in real-world scenarios (both indoors and outdoors) and demonstrate that *AdamVPN* achieves better throughput than OpenVPN over Wi-Fi and LTE environments.

The contributions of this paper are the following:

- We show the benefit of flow-level mobile VPN configuration adaptation, describe why simple solutions (hard resets and multiple tunnels) fail to achieve the same benefit, and identify two key challenges toward achieving flow-level VPN adaptation: (1) mechanism for adapting tunnel parameters seamlessly, and (2) policy for choosing suitable tunnel parameters.
- To the best of our knowledge, our solution is the first to achieve flow-level VPN configuration adaptation while ensuring seamless and uninterrupted network connectivity between the IoT gateway and the cloud.
- We provide an algorithm to choose a suitable VPN configuration from a large number of possibilities, where the choice is based on the type of data traffic, the IoT system and the client's network context.
- We develop *AdamVPN*, which extends COTS SSL-based VPN solution by adding dynamic configuration adaptation capability. With thorough experiments we show that *AdamVPN* achieves 2.75X–3.0X and 1.8x–2.16X more throughput for Wi-Fi and cellular networks, respectively, when compared to that of the OpenVPN.

II. Background

VPN ensures secure end-to-end communication between network nodes over an otherwise insecure network such as the Internet. During the connection setup, VPN creates a *tunnel* between a server and a client. During communication, it encrypts and encapsulates payloads and hands the packets over to the untrusted network.

A. How does an SSL/TLS based VPN work?

Unlike Internet Protocol Security (IPSec) [7], SSL/TLS VPNs run as user applications, is portable and can be developed without requiring modifications to the OS. An SSL/TLS based VPN uses virtual network kernel devices (TUN/TAP) to inject packets to/from the OS and offers many of the key features of IPSec with a relatively lightweight footprint. Figure 1 shows how packets are processed by the SSL/TLS based OpenVPN. A packet enters the kernel via a TCP socket and then is routed back to userspace over the virtual tun0 interface to the OpenVPN application. The OpenVPN encrypts and signs the packet, adds a header, and fragments it (if needed). Finally, a UDP datagram is formed and pushed to the kernel, which then sends the UDP datagram to the Internet. When the packet reaches the other end of the tunnel, the process is reversed.

B. VPN Configuration Parameters

AdamVPN leverages OpenVPN which has more than 180 parameters to configure a tunnel. We have extensively analyzed these and found 54 security related and 23 network throughput related parameters among them.

Parameters, e.g., `keysize` and `crypto-engine`, that are directly or indirectly related to authentication, confidentiality, or integrity are defined as *security sensitive* parameters. For

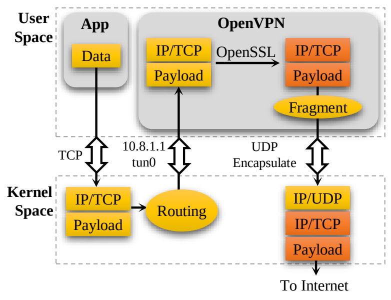

Figure 1: Packet processing in OpenVPN with UDP protocol

instance, the higher the `keysize`, the higher the security strength is. It also has a direct relationship with CPU cycles and network throughput. On the other hand, parameters, e.g., `compression`, and `tun-mtu`, that have direct effects on throughput are defined as *throughput sensitive* parameters.

C. Base Security and Application Default

We assume a *base security configuration* that ensures a sufficient level of security which is presumably unbreakable (e.g., `auth=SHA256`, `cipher=AES`, and `keysize=128`). Given an IoT application's transport security requirements (e.g., cipher and signature algorithms, keysize, replay protection), we translate them to a VPN configuration file containing both security and throughput related parameters for setting up the secure tunnel. We refer to such configuration as *application default* configuration which may vary as different applications set different security levels based on their policies.

III. Motivation and Challenges

A. Flow-Level mVPN Adaptation

mVPN adds overheads to IP packets as it enables secure communication over a VPN tunnel. Depending upon the in-use configuration, a typical mVPN connection's application-level throughput could be several orders of magnitude lower than a VPN-less communication [4]. A commonsense approach to accelerate mVPN is to use a suitably chosen configuration based on the type of the application. Such an *application-level* adaptation may increase an application's throughput to some extent. However, the improvement is still limited since typical IoT gateways [8] send/receive a wide variety of data traffic (e.g., text, audio, video, and sensor values), each having different bandwidth and security requirements. We argue that if, based on the application type, and bandwidth and security requirements of the data traffic, the mVPN parameters are dynamically adjusted at the *flow-level*, the overall throughput of an application can significantly increase with respect to application level adaptation.

In order to understand the benefit of flow-level mVPN adaptation, we conduct an experiment where we measure the achieved throughput (shown in Figure 2) of an IoT gateway under different mVPN settings. We use synthetic data traffic, following the ratio of packets from [9] for Skype, that contains 12% audio, 40% video, and 48% file transfer and text. The data is mixed at Nexus 5 phone used as the IoT gateway and streamlined to an Amazon EC2 server. We choose Skype in this study, since it contains a mix of audio, video, and text which closely emulates the upstream data by an IoT gateway.

We use three different VPN settings– a VPN-less setting, a typical fixed VPN configuration (256-bit key and data compression enabled), and a flow-level adaptive mVPN that adapts its configuration based on the type of a flow of packets. For flow-level adaptation, we use data compression only for text (because, audio-video contents are almost always compressed by the application itself), 256-bit key for only the I-frames containing actual video frames, and 128-bit key for P- and B-frames that are predicted from I-frames. Selective encryption schemes [10]–[12] use similar heuristics for encrypting video traffic to enhance network throughput. Our approach is to use a smaller keysize for P- and B- frames rather than no encryption.

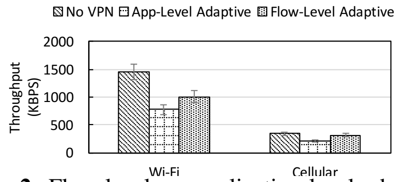

Figure 2: Flow-level vs. application-level adaptation.

Figure 2 plots application throughput for both configurations, for both Wi-Fi and LTE networks in our campus. We observe that a fixed app-level configuration yields 39%-47% lower throughput than the VPN-less case. Flow-level adaptation improves throughput by 48%-50%, by using its knowledge of traffic and network types and switching to a configuration that results in the highest throughput.

B. Overhead of Hard Resets

Once an mVPN connection has been established, if any configuration parameter is changed, in order for the new configuration to be effective, the connection has to reset. This is required because existing mVPN protocols allow a server to load a client-specific configuration file into its memory only at the time of a new connection setup. Although a client is allowed to update the configuration on-the-fly (without restarting the server), the new configuration becomes effective only for new connections and it has no effect on existing connections. Hence, a hard connection reset is the only mechanism to enable a new configuration.

A hard reset is achieved by killing client's instance using the VPN management interface. This causes the client to reconnect and use a new configuration file. During a reset, a new handshake between the client and the server takes place. The process involves exchanging a number of control messages between the mVPN server and its client (total 6449 bytes/reset), and also takes a significantly long time to interrupt an application's on-going network communications noticeably.

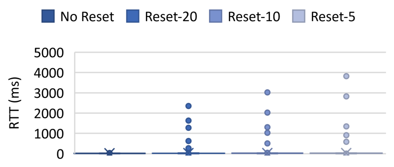

Figure 3: Overhead of hard resets.

Figure 3 shows the overhead of hard resets. We send $1,000$ ICMP packets from an mVPN client to a server, perform periodic forced resets, and measure the round trip times (RTT). The figure shows a box-plot of RTTs where most RTTs are small ($< 30\ ms$), and the large ones (outliers) are the ones that are affected by resets. RTT is as high as $2.35s$ to $3.82s$ when the reset interval is varied from 20 to 5 packets. Since effectively updating the mVPN parameters of an established VPN tunnel based on the flow-type would require even more frequent resets, this is not a feasible approach to achieve flow-level mVPN adaptation due to its high overhead.

#Tunnels	CPU	Memory
1	9.09%	18.29 MB
2	11.64%	36.75 MB
3	15.83%	51.06 MB

Table I: Overhead of using multiple tunnels.

C. Overhead of Multiple Tunneling

An alternative and naïve approach to solve this is to create multiple VPN tunnels – each managed by an individual process and configured with a suitable configuration tailored to a specific type of data traffic. This approach, however, does not scale with the number of traffic classes because creating an individual tunnel for each type of packet imposes substantial CPU and memory overheads on a gateway device. To estimate these overheads, we measure the CPU utilization and the memory footprint of OpenVPN by setting up multiple VPN tunnels between two Raspberry Pis (since smartphones or other COTS gateways do not allow multiple VPN services). Table I shows that both CPU and memory overheads increase somewhat linearly with the number of tunnels. This happens as tunnels are maintained as separate processes which add overheads due to frequent context switches.

D. Faster mVPN via Flow-Level Configuration Adaptation

The fundamental limitations of existing mVPNs and the shortcomings of naïve solutions push the need of – *flow-level mVPN adaptation to achieve a higher throughput by using different mVPN configurations for different types of packets, while at the same time using a single VPN tunnel and avoiding hard-resets*. This is the problem we solve in this paper. We propose an adaptive mVPN solution that we refer to as *AdamVPN*. In the development of *AdamVPN* we faced two major research challenges:

Challenge 1. Devising a *mechanism to update the mVPN configuration* while ensuring an uninterrupted connection and a smooth flow of packets. Any change in the mVPN configuration parameters must not hard reset an existing connection. Rather the new configuration should be effective immediately.

Challenge 2. Devising a *policy for choosing a suitable mVPN configuration* to maintain and/or improve an IoT gateway's network throughput. The challenge lies in segmenting the parameter space formed by a large number of possible mVPN configurations based on their effect on throughput and choosing the most suitable configuration for a given class of traffic, network interface (e.g., Wi-Fi or cellular), current and desirable throughput, and application's security policy.

IV. The Design of *AdamVPN*

A. Adversary Model

We consider a strong adversary model (similar to [4], [7], [13]) where only *AdamVPN* server at the cloud and the client at the gateway belong to the trusted computing base, i.e., the server and client machines are considered to be not compromised through software/hardware based exploitation. We consider the Dolev-Yao [14] attacker model for the connection between the IoT gateway and cloud in which both on-path and off-path adversaries have the capability of injecting unauthenticated packets, modifying legitimate packets, and sniffing end-to-end messages.

B. AdamVPN Overview

AdamVPN solves the two challenges mentioned in the previous section and performs flow-level configuration adaptations based on current data traffic, network, and security contexts. *AdamVPN* extends OpenVPN with two new basic functionalities: selection of a suitable configuration, and adaptation of the selected configuration. These two are referred to as **`Config Selector`** and **`Config Adapter`**, respectively.

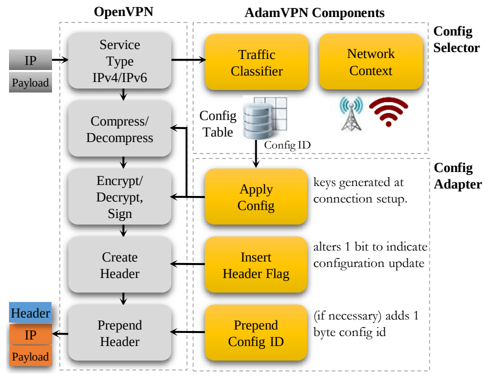

Figure 4: Interactions between OpenVPN and *AdamVPN*.

Figure 4 shows the architecture of *AdamVPN* along with its interactions with OpenVPN components. Typically in OpenVPN, an IP packet goes through five major processing components: (1) IPv4/IPv6 service specific tasks (e.g. altering TCP max segment size), (2) compression, (3) encryption and signing, and (4) creating and (5) prepending a header. Interactions with the **`Config Selector`** and the **`Config Adapter`** change the behavior of some of these components.

The **`Config Selector`** intercepts the IP packets from OpenVPN, uses a **`Traffic Classifier`** to find the packet type, uses information on client's **`Network Context`**, and by combining these two types of information selects a suitable mVPN configuration from a lookup table according to the chosen mode of operation.

The **`Config Adapter`** takes the configuration and applies it by controlling the packet compression, encryption and signing stages, and by adding *AdamVPN* specific information to the OpenVPN-generated header. Further details on **`Config Adapter`** and **`Config Selector`** follows next.

C. AdamVPN Configuration Adapter

The **`Config Adapter`** works in the two phases of an mVPN connection: the initialization of *AdamVPN* happens during connection establishment, and the adaptation of configuration takes place during communication.

1) Initialization: Configurations are likely to be changed from time to time in *AdamVPN*. In order to make the adaptation process efficient, *AdamVPN* precomputes the values of a number of parameters (i.e., encryption/decryption keys for different algorithms and key lengths) at the time of connection establishment. Because these parameters change frequently and have a higher computational overhead, precomputing and storing them at the time of connection establishment makes *AdamVPN* efficient. Because smaller keys are extracted from larger ones, *AdamVPN* requires to negotiate only for the 256-bit (largest) keys during the connection initiation. Since configuration parameters are precomputed as required, they can be loaded and used instantly. Besides saving time, this also eliminates the need for a connection reset, which is mandatory in OpenVPN, whenever an encryption key is required to change.

2) Configuration Adaptation: Once a connection is established, *AdamVPN* enters into the configuration adaptation phase in which the **`Config Adapter`** waits for the **`Config Selector`**'s decision to initiate an adaptation process. Once initiated, **`Config Adapter`** applies the new configuration (by controlling OpenVPN modules), sets an *AdamVPN* specific flag in the OpenVPN-generated header, and prepends the config ID and the signature of config ID so that the other end of the tunnel knows what configuration is in effect.

The **`Apply Config`** component performs two major tasks: controlling the compression (decompression), and encryption (decryption) of packets. If the selected configuration has 'compression' as one of its parameters, e.g., **`comp-lzo`**, the **`Config Adapter`** enables compression (decompression) inside the OpenVPN. Similarly, if a new encryption algorithm/key is mentioned in the configuration, e.g., **`ciper`** parameter, it finds the precomputed key, uses the key to encrypt (decrypt) the payload, and then signs/authenticates it. Note that, for encryption (decryption), OpenVPN peers keep both an active key and a 'lame duck' key (i.e., the last active key prior to a reset). They use one of these keys to process the packet and send the corresponding key ID along with the packet to the remote peer. *AdamVPN* follows the same strategy for the shared keys between peers.

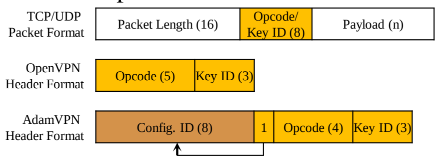

Figure 5: *AdamVPN* uses an unused opcode bit as flag and (occasionally) adds an extra byte to send the newly selected id

The **`Insert Header Flag`** interacts with OpenVPN's header creation process. OpenVPN creates an 8-bit header (Figure 5) to include an opcode (5 bits) and a key ID (3 bits). This header is used by the receiver to identify CONTROL/DATA/ACK packets, and the key is used to process the received packet. Of the 5 opcode bits, one bit is unused by OpenVPN. *AdamVPN* exploits this unused bit to convey the message to the receiving end that the configuration has been changed.

Besides using a flag to indicate a changed configuration, *AdamVPN* needs to pass the new configuration ID to its peer at the receiving end. This is done by **`Prepend Config ID`**, which adds an extra byte containing the config ID to the OpenVPN header. This is added by the sender once per configuration adaptation, and the byte is consumed by the receiver only when the header flag is set.

3) Security & privacy for adaptation: Whenever the configuration adapter requires to change ciphers on-the-fly, *AdamVPN* chooses a new initialization vector to protect against *known-plaintext* attacks. To disable (`lzo`) compression, *AdamVPN* sends the data without compression and does not flush the dictionary.

To protect against packet header forgery attacks, **`Config Adapter`** signs the config ID using HMAC and prepends the signature so that the receiver can detect modified config IDs in case of forgery attacks. Note that, like OpenVPN, *AdamVPN* sends the packet header in plain text (shown with blue color in Figure 4) so that the receiver can determine from the key initially negotiated the key to be used to decrypt the payload.

AdamVPN leverages the data-perturbation technique [15] by mixing the occurrences of actual and fake configuration changes to prevent any fingerprinting. *AdamVPN* adds noisy configuration changing events by sending fake configuration adaptation signals with fake config IDs in the *AdamVPN* header after a random number of packets are transferred. To enable the legitimate clients to differentiate between the actual and fake IDs, *AdamVPN* server shares a unique list of config IDs with each client during the connection establishment phase where actual and fake config IDs are well marked.

D. AdamVPN Configuration Selector

The **`Config Selector`** component runs at the client and decides whether switching to a new mVPN configuration will improve client's throughput. Inside the component, there is a lookup table, referred to as **`Config Table`**, that stores a mapping between a set of mVPN configurations and the corresponding expected client throughputs, given the type of the data traffic and the network context. The type of data traffic is determined by the **`Traffic Classifier`**, and the **`Network Context`** monitoring service provides information on the client's throughput.

1) Traffic Classifier: AdamVPN uses an offline-trained, lightweight, decision tree (C4.5) classifier to classify IP packets into one of the following categories based on their bandwidth: low bandwidth sensor and text data, medium bandwidth files and small images, and high bandwidth multimedia (video with audio), larger files, and images. Frames in video streams are further categorized into three classes: I-, P-, and B- frames since they have different bandwidth and security requirements.

A large amount of training samples were used for traffic classification. The flow of IP packets (either from the Internet or from a content server) is analyzed at the *AdamVPN* server over a window of time to compute up to 30 statistical features. The list of features is shown in Table II. These features are a subset of features found in [16], where they were used to classify a wide variety of Internet traffic types. For a simpler classification task as ours, the first $15-20$ features from Table II are sufficient. With these features, we achieve a classification accuracy as high as 99.1% with an observation window of 5 packets. Since *AdamVPN* does not lower the security level below the *base security* or the *application default*, we ensure that security is not compromised nor reduced for the 0.9% misclassified traffic in the worst case.

Category	Features
Ports	`server port, client port.`
Inter Arrival Time	`min, max, avg, var, 3 quartiles.`
IP Packet	`min, max, avg, var, 3 quartiles.`
Control Bytes	`min, max, avg, var, 3 quartiles.`
Ethernet Packet	`min, max, avg, var, 3 quartiles.`

Table II: The features considered during traffic classification.

Interface Type	Signal Strength	Packet Type	Config ID	Throughput Vector	Throughput Statistics
LTE	A	Video	C_i	$(t_i^1, t_i^2, \ldots, t_i^N)$	$\bar{t_i}$, σ_{t_i}
WiFi	B	Text	C_j	$(t_j^1, t_j^2, \ldots, t_j^N)$	$\bar{t_j}$, σ_{t_j}
...	...	...	...	...	
LTE	A	Video	C_m	$(t_m^1, t_m^2, \ldots, t_k^N)$	$\bar{t_m}$, σ_{t_m}

Table III: **`Config Table`** is shown with three symbolic entries.

2) Network Context: The **`Network Context`** monitor of *AdamVPN* client keeps track of three types of information about the client's connection: first hop wireless network type, throughput estimates, and signal strength level. The type of network, i.e., cellular/Wi-Fi, is determined by the client when the connection is established. *AdamVPN* approximates the throughput by keeping track of the received timestamps of the ACK packets during an ongoing TCP session. The *AdamVPN* client records its Wi-Fi and/or LTE's signal strength periodically (once per minute) and only if there is a significant change in it. It classifies signal strength into one of three categories (A-C) by dividing the full range of signal strength into 3 sub-ranges.

3) Configuration Table: AdamVPN consults a **`Config Table`** as shown in Table III at runtime that contains a configuration to throughput mapping for all combinations of network interfaces, signal strength levels, and packet types. It starts with a default mapping computed off-line and evolves over time. With the help of **`Config Table`**, the **`Config Selector`** (described next) determines if another valid configuration yields a better throughput than what the client is experiencing, given the current packet type and the network context. The search for a valid configuration involves checking constraints on configurations (e.g., based on the mode of *AdamVPN*), and updating the table with new information (e.g., to refine the mapping).

An example of a configuration table with three symbolic entries is shown in Table III. The first four columns correspond to the interface/network type, signal strength class, packet type, and configuration ID, respectively. The last two columns store the last N throughput estimates, and summary statistics (i.e., mean and variance).

4) Configuration Selection and Table Update: AdamVPN provides fine-grained dynamic adaptation at the flow level which allows multiplexing configuration parameters within a single flow. *AdamVPN* performs the following steps to determine a suitable configuration and to keep the entries in **Config Table** updated.

Step 1: Given the network context, packet type, and configuration in use, locate the matching entry i in the table to obtain the mean $\bar{t_i}$ and standard deviation σ_{t_i} of the client's empirical throughput estimates.

Step 2: Compare the current throughput t_c to check if it is within 3 standard deviations of the mean, i.e., $\bar{t_i} - 3.0\ \sigma_{t_i} \leq t_c \leq \bar{t_i} + 3.0\ \sigma_{t_i}$. If not, invoke the configuration adaptation step (Step 3). Otherwise, jump to Step 4.

Step 3: Locate the entry k in the table that has the same network context, packet type, a valid configuration (no conflicts with client's requirement), and the maximum mean throughput $\bar{t_k}$. Select C_k as the new configuration, and notify **Config Adapter**.

Step 4: Insert t_c into the throughput vector of the i^{th} entry after evicting the oldest value. Update the mean $\bar{t_i}$ and the standard deviation σ_{t_i}.

V. Evaluation

We describe the experimental setup, followed by two sets of evaluations: (i) the overhead of *AdamVPN* and the performance of the configuration selection algorithm; (ii) the effect of traffic types and network characteristics on *AdamVPN*.

A. Experimental Setup

1) Devices and Networks: We use a Linux machine in Amazon EC2 to host the *AdamVPN* server (located in Oregon), and configure two Nexus 5 smartphones as *AdamVPN* clients (located in Indiana and California). Since the time required for sensor data transmission from IoT devices to gateways is not in the scope of this paper, we assume that the Nexus 5 phones, working as the IoT gateways, have already cached four types of IoT data: sensor data (variable rate sensor streams from data.sparkfun [17] by specifying its URL), files (plain texts and images), audio (WAVE and MP3 files), and video (3GPP and MPEG-4 files). Also, this setup provides a greater flexibility in evaluating the performance of *AdamVPN* through mixing up different types of traffic for different network contexts at the IoT gateways. We use Wi-Fi networks of a University and an Enterprise research lab, and the T-Mobile's cellular network in Indiana. Each data point is obtained by taking the average of at least 5 executions.

2) Datasets and Applications: For the deployment evaluations described in Section V-D, the Nexus 5 phones acting as gateways mix the data traffic and select the network types as summarized in Table IV.

Traffic Types		Network Types	
D1	Low BW (sensors, text)	Wi-Fi-A	(> -55 dBm)
D2	Medium BW (files < 2 MB)	Wi-Fi-B	([-55, -75] dBm)
D3	High BW (multimedia, 5-18 MB)	Wi-Fi-C	(< -75 dBm)
D4	50% D1, 50% D2	Cellular-A	(5 bars)
D5	50% D1, 50% D3	Cellular-B	(3-4 bars)
D6	50% D2, 50% D3	Cellular-C	(< 3 bars)
D7	33% D1, 33% D2, 34%D3	–	

Table IV: Data and networks used in the empirical evaluation.

B. Overhead Measurement

1) Packet Size Increase: OpenVPN adds extra bytes to an IP packet, onto which *AdamVPN* may add a few more extra bits for config ID. The exact number of extra bits per packet in *AdamVPN* depends on whether or not an adaptation request is sent with the packet. OpenVPN adds a 28 byte IP+UDP header, and based on the VPN configuration, *AdamVPN* adds up to 92 additional bytes to a packet which is 1.04% (Table V) in the worst case when compared to OpenVPN's.

VPN Type	Overhead (bytes)	Increase (%)
OpenVPN	28 + 92 (max)	–
AdamVPN	28 + 92 (no update)	0%
	28 + 97 (update)	1.04%

Table V: Packet size overhead.

2) Packet Processing Time: **Client Side.** To quantify the additional packet processing delay incurred by the *AdamVPN* client, we measure the average processing time of an IP packet as it passes through *AdamVPN*, and compare this time with that of OpenVPN's. We send 10,000 IP packets over an mVPN tunnel multiple times– each time limiting the number of configuration adaptations. Before sending each packet, a biased coin having the probability of success of $\{0.0, 0.25, 0.5, 0.75,$ and $1.0\}$ is flipped to determine if the configuration will be changed for this packet. Note that, we *do* run the packet classifier in this experiment, but its decision is overridden to enable finer control over the adaptation frequency. Hence, the execution time includes the packet classification time. The random numbers are pre-computed; their computation time is not included in packet processing time.

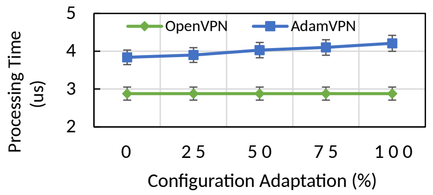

Figure 6: *AdamVPN*'s packet processing time varies linearly from 3.88ms – 4.21ms.

Figure 6 shows how the processing time varies with the frequency of mVPN configuration adaptation. The line denoting OpenVPN remains constant at 2.88 μs as it has no effect on the decision for adaptation. The *AdamVPN*'s packet processing time varies linearly from $3.88 - 4.21$ μs. The worst case (and highly unlikely) packet processing delay in *AdamVPN* is 4.21 μs, which is 46% more than the time of OpenVPN. For such a small increase in packet processing time for *AdamVPN* does not experience any packet loss for such slightly increased packet processing time.

Server Side: The *AdamVPN* server, upon reception of a packet, tests 1 bit of the *AdamVPN* header of the received packet to check for a configuration change. If there is a configuration change, *AdamVPN* consults the Config. Table to determine the new configuration. Both 1-bit test and lookup require a negligible amount of time and thus the packet processing time for OpenVPN and *AdamVPN* servers is practically the same.

3) Configuration Initialization Overhead: During the initial connection establishment *AdamVPN* precomputes the keys for different authentication and encryption/decryption schemes (e.g., `SHA256`, `AES-128-CBC`, `AES-256-CBC`, `BF-CBC`). For the pre-shared secret key mode, *AdamVPN* requires around 1.2 ms more time than OpenVPN to precompute these keys. For SSL/TLS mode, this extra time can be up to 2.3 seconds. However, since precomputing all these keys is a one-time cost, we consider this overhead negligible, especially when compared to the cost of hard resets during communication.

C. Performance of Algorithms

1) Configurations and Throughput Resolution: As discussed in Section IV-D, *AdamVPN* clusters configurations that yield similar throughput in order to improve its efficiency. The goal of this experiment is to quantify how *AdamVPN*'s control over a client's throughput is affected by the number of clusters. To achieve this, we cluster the mean throughput values (one value per config) into k clusters (where, $k \in [50, 500]$) using the **`kmeans`** algorithm, and take the mean of inter-cluster distances to obtain the 'throughput resolution'.

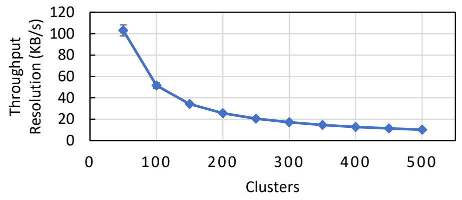

Figure 7: By having 500 configuration clusters, *AdamVPN* reacts to a 10.2 KBps change in throughput.

For example, Figure 7 shows that when we cluster the configurations into 500 clusters, the inter-cluster mean throughput difference is around 10.2 KBps. In other words, *AdamVPN* adapts to a different configuration class when there is about 10.2 KBps rise or drop in a client's throughput.

D. Deployment Evaluation

We have deployed *AdamVPN* in two scenarios: a Wi-Fi environment (indoors) and a cellular environment (outdoors) for deployment evaluation.

1) Effect of Traffic Types on Throughput: AdamVPN's configuration adaptation depends largely on the type of data traffic. In this experiment, we compare the mean throughput at the client (measured throughput as opposed to estimated at the server) for the 7 types of traffic mixes defined in Table IV for *AdamVPN*, OpenVPN, and No VPN.

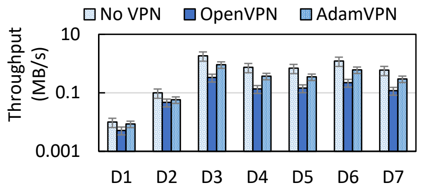

Figure 8: Effect of data traffic types.

We observe that, for any VPN, the achieved throughput is dependent on the type and mix of data (e.g. about 10 KBps for D1 sensor streams, about 1.85 MBps for D3 multimedia traffic, and for others it is in-between these limits). Overall, using OpenVPN reduces the throughput by 72% when compared to No VPN, which is about 43% for *AdamVPN*. *AdamVPN* is 1.23X–2.73X better for low and medium BW traffic, and is 2.4X–2.75X better for traffic that has a significant amount of high BW multimedia data when compared to OpenVPN.

2) Effect of Network Types on Throughput: AdamVPN's configuration adaptation depends on the network context, i.e., the type of the edge network (Wi-Fi and cellular) as well as its signal strength. In this experiment, we compare the mean achieved throughput at the client for *AdamVPN* and the two baselines. We show summarized results for 3 classes of Wi-Fi and cellular networks (A, B, and C) based on their signal strengths.

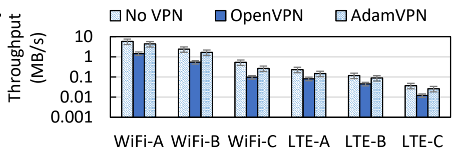

Figure 9: Effect of network type and strength.

Figure 9 shows that a client's achieved throughput with *AdamVPN* is 2.75X–3X higher over Wi-Fi and 1.8X–2.16X higher over a cellular network. The improvement is helpful specifically during slower network connectivity. For example, when OpenVPN's throughput is around 97 KBps on a weak WiFi network, *AdamVPN* improves it to 266 KBps– making it possible to use an application over a secure tunnel, which is not possible otherwise.

VI. Security Analysis and Verification

A. Security Analysis

• *Adversaries cannot inject or modify packets.* Since *AdamVPN* never configures a VPN tunnel without encryption, authentication, or with a weak cryptographic scheme, any maliciously injected or modified packets by adversaries will be always identified.

• *Malicious VPN client cannot identify the traffic type of legitimate clients.* Let the size of a config ID in *AdamVPN* be k-bits, and therefore, the total number of possible IDs is $n = 2^k$. If *AdamVPN* uses only m (where $m < n$) configurations, $(n - m)$ config IDs remain unused– which *AdamVPN* uses as fake config IDs. *AdamVPN* server is able to generate $n!$ number of unique lists of config IDs and allocate one to each client during the connection establishment phase. Therefore, no two clients have the same set of config IDs representing the same configuration parameters as long as the number of clients is at most $n!$. For example, though two clients, c_1 and c_2 are given the same list of config IDs: $\{1, 2, 3, \ldots, 256\}$, config ID 2 for c_1 may represent a fake ID whereas the same config ID 2 for c_2 may represent a valid ID (an ID for a particular client is decided to be fake using a uniform distribution with probability $\frac{n-m}{n}$). Furthermore, config ID 3 for c_1 and c_2 would represent different configurations even if this ID for both clients represents a valid config ID. Thus *AdamVPN* prevents a malicious VPN client from learning the traffic type of other honest clients.

• *Adversaries cannot infer traffic type by fingerprinting config ID changes. AdamVPN* prevents fingerprints by injecting fake config IDs and allocating a new list of config IDs to a client after random intervals. Fake IDs add noises and induce inaccurate analysis of fingerprinting. When there are l clients connected with an *AdamVPN* server, l unique lists are shared with the clients. If $l < \binom{n}{m}$, there will be no collision while reallocating a new list to a client. *AdamVPN* adjusts the values of n accordingly to avoid such collisions.

B. Verification

We first model the proposed communication protocol between the IoT gateway and the cloud server using a cryptographic protocol verification tool ProVerif [18] and then formally verify the following properties: (1) secrecy of messages and config IDs; (2) authenticity of config ID in *AdamVPN* header with injective queries; (3) observational equivalence of two config IDs sent by two clients. We observed no counterexample against our proposed mechanism that violates the given properties.

VII. Related Work

Despite heavy uses of VPNs, few research efforts [4], [13], [19]–[23] have been devoted to improving SSL-based user space VPNs. [20] investigates the effect of encryption and authentication methods for OpenVPN. [4] proposes a flexible tunnel approach where customized VPN functions are applied to applications. [13] proposes a secure communication mechanism for IoT using lightweight IPsec [7], DTLS, and 802.15.4 link layer security. Unlike *AdamVPN*, they do not consider the dynamic nature of networks, applications, and traffic, and do not address the issue of hard resets.

[24] provides a transport layer mVPN for 3G and WLAN air interface technologies. [21] develops a virtual network service with custom management capabilities to support VPN level QoS. [22] designs a VPN leveraging network layer peering, circuit switching, and link layer circuit and per stream switching. [23] designs a protocol stack and mandatory VPN services which can be used for OSI architecture, ATM architecture, and over WDM optical networks. VPMN [25] proposes a virtualization mechanism to dynamically create private, resource isolated, customizable, end-to-end mobile networks. [26] models traffic loss to maximize carried traffic in VPN. These approaches may be applied to a specific type of network, but are not sufficient to handle the dynamic nature of mobile networks and mobile data patterns.

VIII. Conclusion

We have proposed, *AdamVPN*, which is a flow-level adaptive mobile VPN that adapts its configuration at runtime to improve the next generation IoT gateway's application-level throughput based on current data traffic and network context. We propose two novel algorithms: selection of a suitable configuration and dynamic adaptation of the selected configurations. *AdamVPN* has been shown to achieve higher throughput than the OpenVPN and also close to the VPN less solution for both Wi-Fi and cellular networks.

Acknowledgment

This paper was supported, in part, by NSF grants CNS-1816213 and CNS-1704469.

References

[1] *New IoT Threat Exploits Lack of Encryption in Wireless Keyboards*, https://www.esecurityplanet.com/network-security/new-iot-threat-exploits-lack-of-encryption-in-wireless-keyboards.html.

[2] M. Kassner, *No surprise, IoT devices are insecure*, http://www.techrepublic.com/article/no-surprise-iot-devices-are-insecure/.

[3] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic," *CoRR*, vol. abs/1705.06805, 2017.

[4] S. Khanvilkar and A. Khokhar, "Virtual private networks: an overview with performance evaluation," *Communications Magazine, IEEE*, vol. 42, no. 10, pp. 146–154, Oct 2004.

[5] J. Leyden, *90% of SSL VPNs are hopelessly insecure, say researchers*, https://www.theregister.co.uk/2016/02/26/ssl_vpns_survey/.

[6] L. NEWMAN, *BEWARE: MOST MOBILE VPNS AREN'T AS SAFE AS THEY SEEM*, https://www.wired.com/2017/02/beware-mobile-vpns-arent-safe-seem/.

[7] N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtua*. Prentice Hall PTR Publishers.

[8] *Intel IoT Gateway Technology: Wind River IDP XT 3.1 Getting Started Guide*, https://software.intel.com/en-us/Setup-IDP-DevelopmentTools.

[9] P. del Rio, D. Corral, J. L. García-Dorado, and J. Aracil, "On the impact of packet sampling on skype traffic classification," in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013),*, 2013, pp. 800–803.

[10] I. Agi and L. Gong, "An empirical study of secure mpeg video transmissions," in *Proceedings of the 1996 Symposium on Network and Distributed System Security (NDSS '96)*.

[11] T. li Wu and S. F. Wu, "Selective encryption and watermarking of mpeg video (extended abstract)," in *the International Conference on Image Science, Systems, and Technology, CISST, Las Vegas*, 1997.

[12] G. Spanos and T. Maples, "Performance study of a selective encryption scheme for the security of networked, real-time video," in *4th Int. Conf. on Computer Communications and Networks*, 1995, pp. 2–10.

[13] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing Communication in 6LoWPAN with Compressed IPsec," in *7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Barcelona, Spain, Jun. 2011.

[14] D. Dolev and A. C. Yao, "On the security of public key protocols," in *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*, ser. SFCS '81, 1981, pp. 350–357.

[15] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the privacy preserving properties of random data perturbation techniques," in *Third IEEE International Conference on Data Mining*, Nov 2003, pp. 99–106.

[16] S. Kaoprakhon and V. Visoottiviseth, "Classification of audio and video traffic over http protocol," in *9th International Symposium on Communication and Information Technology.*, 2009, pp. 1534–1539.

[17] *Data.Sparkfun*, https://data.sparkfun.com/.

[18] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW), 2001.*

[19] Z. M. Mao, D. Johnson, O. Spatscheck, J. E. van der Merwe, and J. Wang, "Efficient and robust streaming provisioning in vpns," in *Proceedings of the 12th International Conference on World Wide Web.*

[20] B. Hoekstra and D. Musulin, Tech. Rep.

[21] L. K. Lim, J. Gao, T. S. E. Ng, P. R. Chandra, P. Steenkiste, and H. Zhang, "Customizable virtual private network service with qos," *Comput. Netw.*, vol. 36, no. 2-3, pp. 137–151, Jul. 2001.

[22] H. Lee, J. Hwang, B. Kang, and K. Jun, "End-to-end qos architecture for vpns: Mpls vpn deployment in a backbone network," in *International Workshops on Parallel Processing.*, 2000, pp. 479–483.

[23] K. H. Cheung and J. Mišic, "On virtual private networks security design issues," *Comput. Netw.*, vol. 38, no. 2, Feb. 2002.

[24] D. Benenati, P. M. Feder, N. Y. Lee, S. Martin-Leon, and R. Shapira, "A seamless mobile vpn data solution for cdma2000, umts, and wlan users," *Bell Labs Technical Journal*, vol. 7, no. 2, pp. 143–165, 2002.

[25] A. Baliga, X. Chen, B. Coskun, G. de los Reyes, S. Lee, S. Mathur, and J. E. Van der Merwe, "Vpmn: Virtual private mobile network towards mobility-as-a-service," in *Second International Workshop on Mobile Cloud Computing and Services.*

[26] H. Lian and A. Faragó, "Optimizing virtual private network design using a new heuristic optimization method," *Computer Networks.*