(Self) Driving Under the Influence: Intoxicating Adversarial Network Inputs

Roland Meier[1], Thomas Holterbach[1], Stephan Keck[1], Matthias Stähli[1],
Vincent Lenders[2], Ankit Singla[1], Laurent Vanbever[1]

[1]ETH Zürich, [2]armasuisse

ABSTRACT

Traditional network control planes can be slow and require manual tinkering from operators to change their behavior. There is thus great interest in a faster, data-driven approach that uses signals from real-time traffic instead. However, the promise of fast and automatic reaction to data comes with new risks: malicious inputs designed towards negative outcomes for the network, service providers, users, and operators.

Adversarial inputs are a well-recognized problem in other areas; we show that networking applications are susceptible to them too. We characterize the attack surface of data-driven networks and examine how attackers with different privileges—from infected hosts to operator-level access—may target network infrastructure, applications, and protocols. To illustrate the problem, we present case studies with concrete attacks on recently proposed data-driven systems.

Our analysis urgently calls for a careful study of attacks and defenses in data-driven networking, with a view towards ensuring that their promise is not marred by oversights in robust design.

ACM Reference Format:
Roland Meier, Thomas Holterbach, Stephan Keck, Matthias Stähli, Vincent Lenders, Ankit Singla, Laurent Vanbever. 2019. (Self) Driving Under the Influence: Intoxicating Adversarial Network Inputs. In *Proceedings of The 18th ACM Workshop on Hot Topics in Networks (HotNets'19).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3365609.3365850

1 INTRODUCTION

"Blue boxes" were popular electronic devices in the 1960s which enabled to perform free long-distance calls by abusing the phone switching system. To do so, these devices were simply replicating the audio tones required to start and end calls, as well as the ones allowing long-distance connections. The phone companies ultimately became aware of these abuses and decided to switch from in-band-signaling, which merges the data and the control channels together, to out-of-band-signaling, which separates the two.

Thus far computer networks have been essentially spared by "in-band-signaling" attacks since they, too, rely on out-of-band-signaling. Indeed, the behavior of the control plane (e.g., to decide how to forward the traffic) does not depend on the actual traffic being forwarded in the data plane. This means that to influence the decision of a router, an attacker first needs to access its control plane. While this is possible—BGP hijacks are just one example—this separation heavily reduces the control plane's attack surface. Clearly, not everyone has access to BGP-speaking routers.

Decoupling the data- and the control planes is not "all roses" though. It also leads to suboptimal network behavior such as slow convergence or suboptimal routing: control planes can happily direct traffic into blackholes for hundreds of seconds upon failures, or send all the traffic along suboptimal paths while vastly superior ones are left idle. Frustrated by this and motivated by the emergence of programmable data planes, many researchers have been arguing that network control should be made traffic-aware. Among others, such *data-driven* networks [1, 10, 15, 27, 28, 41, 58, 59] have been proposed for: optimizing routing strategies [54]; quickly reacting to failures [23]; improving traffic engineering [58], streaming quality [39], or throughput in general [13].

While useful, we show that these systems are prone to fake signals, just like the phone system was back in the 1960s. Worse yet, we show that performing these attacks is both easier, and can have much more far-reaching consequences than placing a few free calls. For some attacks, sending crafted packets from a *single* host is enough to trick the system. Likewise, by (maliciously) inducing, say rerouting events, attackers can negatively impact the behavior of a large portion of the traffic, and hence a large portion of users.

Thus, if we want data-driven network systems to be widely-deployed—which they should be, given the amount of exciting applications—we (as a community) must be able to: *(i)* precisely understand their attack surface; *(ii)* formulate and reason about the attacker model; and *(iii)* protect them from adversarial inputs.

HotNets'19, November 14-15, 2019, Princeton NJ, USA

ACM ISBN 978-1-4503-7020-2/19/11...$15.00
https://doi.org/10.1145/3365609.3365850

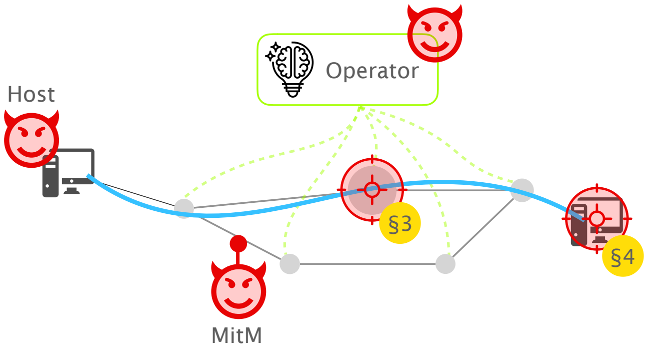

Figure 1: Three types of attackers with different privileges (host, man in the middle and operator) and two targets (network infrastructure and endpoints).

We start to answer these research questions in this paper and make three main contributions.

First, we characterize the threat model (§2). We consider two dimensions: the attackers (how powerful are they?), and the targets (the network? the endpoints?). We believe that this threat model can be expanded upon and used as a blueprint for analyzing data-driven network systems.

Second, we present concrete attacks on recent data-driven network systems, including Blink [23], Pytheas [29], and PCC [13]. Each of these attacks is a representative of a different class of attack techniques, which generalize to a wide variety of systems. For Blink, we show that a single host can induce undesired, and possibly detrimental, rerouting events. For Pytheas, we show how attackers can, by faking performance reports, negatively impact the decisions made for a large set of clients. Finally, for PCC, we show that by manipulating packets, attackers can make its algorithm ineffective, and exploit it to cause traffic fluctuations at a target.

Third, we outline a set of broad strategies for countermeasures (§5), which we hope will inform a substantial branch of the community's research in this direction.

The inspiration for our effort stems from recent work on adversarial inputs in other communities. For instance, adversarial examples—inputs that result in incorrect outputs—are a well-known problem in machine learning applications [11, 21, 25, 57]. Specifically, neural networks have been proven to be susceptible in a way that slight modifications of a benign input can result in a wrong output [3]. Examples include self-driving cars that are tricked by small stickers on street signs [14] and facial recognition systems that are tricked by crafted eyeglass frames [50]. Despite these fields of study being far separated from networking, the techniques for attacking and defending such systems share similarities. Thus, as interest in data-driven networking increases, we should use the experience gathered elsewhere to flesh out potential attack vectors and defenses. In this paper, we take the first steps towards this goal.

2 THREAT MODEL

Who might attack **what** in a network using adversarial inputs? In this section, we model the threat and summarize a (surely incomplete) view of these aspects in Fig. 1.

2.1 Attacker privileges

We consider three different levels of privilege an attacker may have: host, man in the middle, and operator. For all the attackers, we follow Kerckhoff's principle [45] and assume that they know everything about a system (e.g., code and parameter values) except secrets such as cryptographic keys.

Host(s) This attacker has compromised one or more hosts and can manipulate traffic that these hosts send or receive, including being able to inject traffic from such a host.

Man in the middle (MitM) This attacker has intercepted one or multiple links in the network. She can record, modify, drop, and delay traffic that crosses these links, as well as inject traffic. However, she cannot break encryption.

Operator This is the most powerful attacker, with full control over the network. She can record, modify, drop, delay and inject traffic at any location in the network. Furthermore, she can manipulate the network configuration. For example, this privilege level may be obtained by phishing or social engineering of a benign operator.

2.2 Attack targets

Attackers may target the infrastructure or endpoints.

Network infrastructure (§3) These attacks target devices that forward traffic (e.g., routers). Data-driven networks—as we address them in this paper—rely on data-plane signals to take decisions (e.g., to change the forwarding). Typical signals are values in packet headers (e.g., TCP sequence numbers), metadata (e.g., timing) or contents.

An attacker can manipulate the forwarding behavior of a data-driven system by deceptive signaling. Especially, the move from simple, stateless data planes towards complex stateful ones (e.g., using P4 [6]) greatly expands the attack surface. Such manipulations can deteriorate performance (e.g., by sending traffic via a low-bandwidth link); compromise privacy (e.g., by sending traffic through an eavesdropped path); or prepare the ground for other types of manipulation.

Endpoints (§4) These attacks target endpoints and applications running on them (e.g., streaming, file transfer, congestion control). Those applications typically trust the data that they receive from the network. Although a large share of the traffic is encrypted nowadays, it is easy to manipulate the *metadata* of packets (e.g., headers) as well as the traffic statistics (e.g., delay). Manipulating packets or traffic statistics can have an impact on an application's or protocol's performance (e.g., if an adversary manipulates TCP window size) or the situational awareness of users (e.g., if an adversary manipulates information in debugging tools such as traceroute).

3 ADVERSARIAL INPUTS TO DATA-DRIVEN NETWORKS

With researchers arguing for data-driven networks, more and more network decisions rely on traffic, especially after the recent progress in data-plane programmability. As the data plane and the control plane are often considered as the 'thinking fast' and 'slow' components in networking respectively[1], eschewing the control plane's careful-but-slow decision-making in favor of the data plane's speed is tempting. However, this can also make data-plane decisions vulnerable in case tasks usually tackled by the control plane (e.g., routing [43]) are performed in the data plane. This is a big risk because the data plane's inputs are not tightly controlled like the control plane's: packets drive the data plane.

Two components determine the output of a data-driven system and constitute the **attack surface**: *algorithms* that decide which action to take based on the traffic, and their *state*. Manipulating *algorithms* requires *operator* privileges, while *state* can be manipulated by *hosts* or *MitM* attackers. Possible **impacts** of attacks against the forwarding behavior of a network include *(i)* privacy (if the attacker can hijack traffic); *(ii)* performance (if traffic is sent along longer or congested paths); *(iii)* reachability (if the network is disconnected); and *(iv)* revenue loss (if clients observe poor performance).

In this section, we explore the effect of adversarial inputs and how they can negatively affect network decisions. We start with a case study based on Blink [23], a system to quickly reroute traffic upon network failures, where we show that adversaries can fake link failures and hijack traffic (§3.1). Afterwards, we show that many other data-driven network applications are also vulnerable to adversarial inputs (§3.2).

3.1 Manipulating Blink

Blink [23] tries to detect and avoid failed paths without waiting for BGP withdrawals. It infers connectivity disruption by detecting an increase in retransmitted TCP packets. While this significantly reduces the time to detect failures, it also enables attackers to manipulate Blink's decisions by faking TCP signals, as we show in the following.

Blink runs in programmable network devices and monitors a small sample of flows (e.g., 64) for each destination prefix. If half of these monitored flows retransmit packets, it infers a failure and reroutes this prefix along a different next-hop. To choose the monitored flows, Blink computes a hash of each flow's 5-tuple and uses the hash value as an index in an array of cells. Therefore, several flows may collide in one cell. However, at any given time, only one flow occupies a cell, and is thus monitored. This monitored flow is evicted by freeing its cell if it finishes or becomes inactive for 2 s or more. When a cell is free, Blink samples a new flow. Blink also resets its monitored sample every 8.5 min. Every monitored flow is thus eventually evicted, even if continuously active.

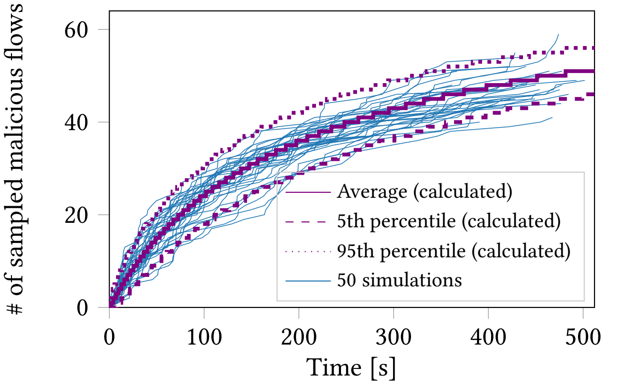

Figure 2: Malicious flows sampled by Blink over time ($t_R = 8.37s$, $q_m = 0.0525$). On average, it takes 172 s until the sample contains enough (i.e., 32) malicious flows.

Attack Manipulating Blink may seem trivial since generating TCP retransmissions is easy. While this is an essential ingredient, Blink's sampling strategy necessitates for more care. The attacker needs to generate flows that always remain active, so that once Blink samples a malicious flow, it keeps monitoring it. Thus, the number of sampled malicious flows increases over time, until the sample is reset. Below, we show that an attacker can often ensure that her flows are the majority of the sampled flows for a prefix. Once this is the case, the attacker can easily trick Blink into rerouting traffic, possibly onto a path that she controls. In practice, one can perform this attack by sending the fake TCP retransmissions from a set of hosts that reach the victim prefix via Blink. Observe that the attacker does not need to establish TCP connections with the victim network, which makes the attack easier to setup and harder to prevent.

Theoretical analysis Let t_R be the average time a legitimate flow remains sampled. We assume a malicious flow is always active, and thus once being sampled, it is never evicted unless the sample is entirely reset. We define t_B as the frequency at which Blink resets the sample ($t_B = 8.5$ min, by default). Thus, t_B is the attacker's time budget until all the sampled flows are evicted. We define q_m as the fraction of traffic that is malicious. For a particular cell of the array used for sampling, the probability p that it is occupied by a malicious flow at the end of the time budget t_B is $p = 1-(1-q_m)^{(t_B/t_R)}$. Now, consider X as a random variable corresponding to the number of malicious flows monitored across all cells at the end of the time budget, t_B. As each of the n cell acts independently, X is binomially distributed with parameters n and p. We use this distribution to calculate the practicality of the attack.

Fig. 2 shows the number of monitored malicious flows over time. We consider $t_R = 8.37$ s, the value computed for one prefix of a CAIDA trace [8] used in Blink's analysis, and $q_m = 0.0525$ (i.e., 5.25 % of the flows are malicious). After

[1] With apologies to Daniel Kahneman [30]

200 *s*, there is a high chance that at least 32 monitored flows (i.e., half) are malicious (purple lines), enabling a successful attack. With longer t_R, the attack is harder, i.e., requires higher q_m. We analyzed the top-20 prefixes of each CAIDA trace used in [23] and found that for half of them the average time a flow remains sampled is 10 *s* (the median is ∼5 s). The example in Fig. 2 is therefore representative. While 5 % of traffic to a destination is substantial, it is within reach, even for the most popular destinations, using a small botnet.

Experimental results To confirm our theoretical results, we simulated a network with mininet [34] and the P4_{16} implementation of Blink. We generated 2000 legitimate and 105 malicious flows ($q_m = 0.0525$), and used the same $t_R = 8.37$ *s*. The thin blue lines in Fig. 2 show the results of each experiment. As expected from the theoretical results, half of the sampled flows are malicious after ∼200 s. We did experiments with many values for t_R and the results always match the theoretical analysis. This confirms that an attacker can manipulate Blink *quickly* and with a *small* amount of traffic.

3.2 Attacking other systems

While we concretely flesh out the attack on Blink, it is by no means the only vulnerable system. Below, we list a set of routing systems running in the control- and data plane which are vulnerable to adversarial inputs.

Control-plane programs Several systems [2, 5] use data-plane signals to improve path selection. For instance, *RON [5]* is an overlay network which reroutes traffic from one node to another when it detects a performance degradation. An attacker in the path between two nodes could drop or delay RON's probes, so as to divert traffic to another next-hop.

Google Espresso [60] and *Facebook EdgeConnect* [49] use passive measurements to extract information and send traffic on the best-performing path. An attacker could lower the performance (e.g., increase the delay) of the flows destined to these networks so that they use another path.

Data-plane programs Several systems similar to Blink have been proposed [24, 42, 43], and we expect similar vulnerabilities to arise in these contexts. *DAPPER [18]* relies on TCP information to determine if a connection is limited by the sender, the network, or the receiver. An attacker can implicate either of these three for performance problems by manipulating TCP packets, and falsely trigger the recourses suggested by the authors.

SP-PIFO [4] approximates PIFO behavior using the strict-priority queues available in programmable switches. The proposed heuristic is based on the assumption that given a rank distribution, the order in which packet ranks arrive is random. An attacker could send packet sequences of particular ranks, resulting in packets being delayed or even dropped.

Some existing data-plane applications also use a number of states that scale according to the traffic (e.g., *SilkRoad [42]* maintains per-connection state). As programmable switches have limited memory, these applications are more vulnerable to DDoS attacks than their software-based counterparts.

FlowRadar [36] and *LossRadar [35]* use probabilistic data structures such as bloom filters to monitor network performance. These data structures are vulnerable against adversarial inputs [12, 17] because they are often dimensioned for the average case, rather than the worst case. An attacker can pollute, or even saturate (cf. [17]) a bloom filter, resulting in inaccurate network statistics.

Recently, Siracusano et al. [51] have shown how to run the forward pass of a binary neural network in the data plane. While promising, neural networks are vulnerable to adversarial examples [3], and thus are particularly exposed in a setting where anyone can inject inputs over the Internet.

Recent work also offloads a set of compute operations from hosts into the data plane [43, 46, 47]. This also opens up new doors for potential attacks, as switches are likely to be shared among users, whereas applications are designed to run in isolated environments.

4 ADVERSARIAL INPUTS TO ENDPOINTS AND APPLICATIONS

In addition to network-based systems, many endpoint-based systems also rely on network traffic to (locally) optimize throughput, make better forwarding decisions or monitor the network performance. A well-known example is TCP and its congestion control algorithm [26].

As for network decisions though, relying on network traffic makes endpoints vulnerable to adversarial network inputs.

The possible **impacts** include *(i)* security and privacy issues (e.g., if addresses are changed such that traffic goes to the wrong destination); *(ii)* loss of situational awareness (e.g., if probing tools are manipulated); *(iii)* performance loss (e.g., manipulated window size in TCP) and; *(iv)* broken debugging tools (e.g., if ICMP packets are manipulated).

Each of the three **attacker models** (cf. §2) can feed adversarial inputs to endpoints and their applications: An *attacker with host-privileges* can only provide inputs to compromised hosts or inject packets to other hosts; an *attacker with MitM-privileges* can provide adversarial inputs to all endpoints which communicate over the compromised links; and an *attacker with operator-privileges* can provide adversarial inputs to all endpoints in the network.

To illustrate the threat, we describe three case studies: First, we analyze Pytheas [29], a framework to optimize Quality of Experience (QoE), and show that malicious clients can influence the system such that other clients receive worse video-streaming quality (§4.1). Second, we show how the throughput of PCC [13], a transport protocol, can oscillate due to adversarial inputs (§4.2). Third, we describe how NetHide [40], a DDoS defense system, can be used to present wrong network topologies to users (§4.3).

4.1 Manipulating Pytheas

Pytheas [29] is a framework to optimize Quality of Experience (QoE), for example for video streaming. Pytheas performs data-driven QoE optimization through a real time exploration and exploitation (E2) process. The driving signals are QoE measurements reported by individual clients, which are grouped by their session similarity (e.g., hosts in the same ISP or location). The E2 algorithms run on group granularity, allowing Pytheas to scalably optimize traffic in real time.

Attacking Pytheas Since decision-making happens at group granularity, if multiple clients within a group report manipulated QoE measurements, this can drive decisions for other clients. For instance, a botnet can pollute measurements for a group of clients seeing video streams by reporting low throughput and poor QoE, such that the system lowers video quality for all clients in the group. *MitM* attackers can achieve similar outcomes if they drop packets for a subset of the group members. In many settings, group membership will not be hard to ascertain even for external parties, as it is typically based on features like autonomous system, IP prefix and location.

An *operator* with deep control of multiple devices can—of course—make such attacks more powerful. An attacker with *operator-level privileges* can program the data-plane hardware to identify traffic of interest, and reduce its throughput, increase loss, and even increase latency by either sending packets along longer paths or bouncing them back-and-forth between devices.

Another possible attack with *MitM* or *operator* privilege is to throttle user flows to/from a particular content distribution network (CDN) site, while prioritizing traffic to others. This way, the attacker can create imbalance and potentially overload one site as entire groups of clients switch to it.

Note that both of these attacks require tampering with only a small fraction of traffic to cause disproportionate damage, by exploiting the group-based decision logic. For Pytheas-like systems, this poses a challenge because other than the observed performance differences, there are no features that separate clients that are performing well from those that are not, forcing a group-wide decision that is potentially harmful, such as lowering the served video resolution. What makes these attacks even more damaging is the prospect that clients that observe poor performance a few times will use the service less (potentially causing revenue loss for the service provider) or worse, even abandon a service in favor of its competitors. Google, for instance, observed that users that saw substantially increased Web search response latency not only made fewer searches, but that this effect persisted even after latency returned to normal [7]. Thus, a short-term active attack can have long-term negative consequences.

4.2 Causing PCC to oscillate

PCC [13] is a novel transport protocol that takes a data-driven approach to congestion control instead of using hard-coded rules as in TCP (e.g., cut rate by half on loss). PCC does A/B experiments with the transmission rate, trying rates that are larger or smaller by a factor of ϵ than the current one. It measures a utility function defined in terms of loss and throughput to pick a direction of movement, i.e., ϵ fraction larger or smaller rate.

By tracking PCC flows, a *MitM* attacker can try to ensure that they see the same utility with both larger and smaller rates. In each monitoring interval, PCC does a rate experiment. This monitoring interval can be estimated from the RTT, which is easy to track in the data plane by programmable network devices.

Knowing the utility function, the attacker can drop packets in the $+\epsilon$ and $-\epsilon$ phases, such that PCC is unable to see a large-enough utility difference. PCC then repeats its experiment with increasing ϵ until a threshold of 5%. Thus, the attacker can cause PCC flows to fluctuate by ±5%, without allowing them to converge to the right rate. Further, by doing this across a large number of PCC flows towards the same destination, the attacker can create sizable traffic fluctuations at the destination, causing challenges with managing this variable traffic.

Not only is PCC's logic neutralized in this setting, it is effectively a tool for the attacker to cause disruption at the destination.

4.3 Faking network topologies

Traceroute is widely used for debugging network connectivity issues. It sends a series of IP packets with increasing time-to-live (TTL) values, and receives the ICMP time exceeded messages from the routers where these TTLs expire. From the source addresses of these replies, it reconstructs the path that packets take.

Since there is no authentication of these ICMP replies, any attacker who can manipulate them can control the path that traceroute displays and thus the topology which the user learns. To perform this attack, it is enough to rewrite the source address of the ICMP replies or to reply to IP packets directly (i.e., craft the entire ICMP replies).

NetHide [40] uses this technique to manipulate traceroute packets such that attackers cannot identify parts of the topology which are susceptible to a DDoS attack. While the focus of NetHide is to use this technique for defense purposes (NetHide limits the amount of lying to the minimum that is required to meet the security requirements), the exact same technique could be used by malicious operators to present wrong information about the topology.

5 COUNTERMEASURES

There is clearly value in extracting actionable network control information from data-plane signals, but for such systems to be practical, we must guarantee that this does not compromise security. We thus describe possible countermeasures to prevent malicious behavior in data-driven systems and specify open questions for future research.

At a high level, we propose to extend data-driven systems by external supervisors, which monitor the systems and prevent them from misbehaving. We illustrate our proposed architecture in Fig. 3, drawing on a self-driving car analogy. A *driver* drives the network while a *supervisor* supervises the driver and determines the directions in which the it can move. The key idea is to not rely solely on data-plane signals but to have an additional feedback loop that checks the plausibility of the signals and controls system. In this framework, one can apply countermeasures at many points: the inputs to the system (I); its code and configuration (II, V); as well as its outputs and behavior (III, IV). We next discuss some possible countermeasures and how these would safeguard the applications we discussed previously.

I Ensuring input quality Most attacks we outlined stem from data-driven applications blindly trusting the inputs they receive, and the inputs not being protected against manipulation. Possible countermeasures in this area include *(i)* encrypting and/or authenticating inputs; *(ii)* improving input quality by using many independent inputs; or *(iii)* verifying inputs, for example through active probing.

However, all these measures are challenging to implement: *(i)* encryption and authentication would require changes in today's prevalent protocols and the required cryptographic operations are not available in today's programmable data planes; *(ii)* deciding based on multiple signals is not possible for every application and it is difficult to assess whether two signals are truly independent and no adversary can manipulate both of them; *(iii)* verifying signals increases the decision time and thus conflicts with immediate reactions to events, which is often desired in data-driven applications.

Research question Where is the sweet spot for maximizing input quality given the cost of modifying existing protocols, modifying applications, and impact on decision time?

II Testing and verifying program code To prevent attacks in the first place, testing and verifying the code of data-driven systems seems promising. However, these are challenging problems. As an illustration, verifying stateful (data-driven) network functions is known to be undecidable [55]. We propose to explore the following directions. For testing, one can use fuzzing techniques that enable autogeneration of (realistic) adversarial inputs (e.g., from existing traces). For verification, we suggest to leverage recent advances in symbolic execution to capture and analyze the behavior of data-driven systems. More particularly, one can capture the semantics of data-driven systems as formulas for SMT solvers and rely on symbolic execution to find sequences of packets (or lack therefore) that could trigger insecure behaviors. Existing works (cf. §6) use a similar technique to find bugs in P4 programs. However, these works are limited to single packets and focus on finding known bugs. A promising strategy that uses a similar approach for automated synthesis of malicious inputs would be to extend the symbolic execution to multiple packets and stateful programs. Given a specified part within a data-plane program (e.g., a register access), such a tool could compute inputs (i.e., sequences of packets) that reach this part of the program. Kang et al. [31] present early work in this direction.

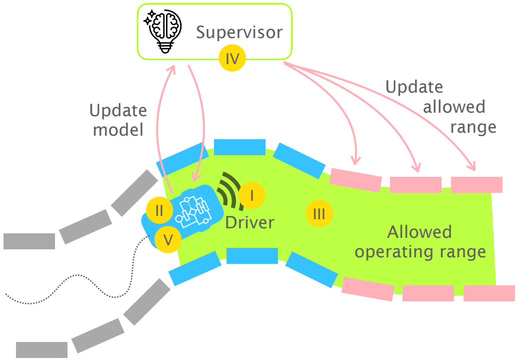

Figure 3: Countermeasures can be applied at many different points of a data-driven system.

Research question How can we adapt and extend existing works on automated program testing and verification for applications in self-driving networks?

III Constraining the decision range of the driver Assuming there exists a model which describes *normal* behavior of a network with respect to the driver application, one can *(i)* compare the behavior of the driver with the model and determine whether it drives "under the influence" (i.e., receives adversarial inputs); *(ii)* use this model to constrain the behavior of the driver proactively. That is, the driver is only allowed to change its state as far as the model allows.

Research question How can we model plausible behaviors of a network and constrain the driver accordingly?

IV Invoking supervisor checks A good supervisor needs to fulfill two criteria: *(i)* it needs to be able to prevent (or at least detect) adversarial inputs; and *(ii)* it needs to do this without impact on the driver's performance.

To satisfy *(i)*, the supervisor needs a model of (non-)adversarial behavior (see III) and ways to compute it. However, this complex modeling and computation is difficult to perform in the data plane at line rate (i.e., without violating *(ii)*) given the limited resources. Thus, while the driver operates in the data plane, we suggest that it invoke a supervisor outside the data plane for safety checks.

The driver determines its current state (e.g., the congestion in the network) and sends this information to the supervisor.

The supervisor then uses a model of plausible states to estimate the risk of the driver being under influence, i.e., being fed adversarial inputs. Then, the supervisor computes the directions in which the driver can steer the network in the future (i.e., possible future states) and sends this to the driver. Always requiring synchronous driver-supervisor interaction will slow down the driver, impacting its effectiveness. But what types of decisions a driver can take asynchronously (without waiting on the supervisor in real-time) may require driver- and decision-specific analysis.

Research question How does an efficient driver-supervisor interface look like, and how do we trade off fast, asynchronous operation against delays in enforcing safety?

V Obfuscating control logic Successful attacks require a model of the control logic used in a data-driven system. Obfuscating this logic, or varying it over time, can thus hinder attacks. This security-by-obscurity method, while less preferable to the other methods discussed above, can form part of a defense-in-depth approach.

Research question: How can control logic, configuration, and state of self-driving networks be obfuscated in a way that makes it hard for an attacker to invert but does not degrade performance and functionality?

Applicability to Blink, Pytheas and PCC We briefly sketch how some of our countermeasures could help protect the systems that we discuss in this paper.

Blink could monitor the RTT distribution over a large number of flows, approximate the expected RTO distribution upon a failure, and use it to distinguish between actual failures and malicious events. Manipulating Blink would then require an attacker to know the RTT distribution of the legitimate flows forwarded by the Blink router, information that is hard to obtain for an attacker with *host* or *MitM* privileges.

Pytheas could look at the distribution of throughput across all clients in a group. If only a few clients exhibit low throughput while others exhibit high throughput, this is indicative of either groups being ill-formed or malicious inputs from part of the group population. Accordingly, the low-throughput clients can be tackled separately, removing their impact on the larger population.

PCC could monitor when packets are dropped in every $+\epsilon$ or $-\epsilon$ phase as well as limit the amplitude of the oscillations by decreasing the range of ϵ.

6 RELATED WORK

In this section, we discuss work on adversarial inputs in other areas (e.g., machine learning), on analyzing data-plane programs and similar attacks.

Adversarial examples Adversarial examples are a well-known and extensively addressed problem in machine learning applications, especially computer vision (cf. survey in [3]). In contrast to these works, we focus on the special circumstances of data-driven systems in networks and do not restrict our focus to machine learning applications. Furthermore, there exists work on adversarial examples for machine-learning based network applications (e.g., malware classification [21, 25, 53, 57] and intrusion detection [11]).

Data-plane software analysis Initial work on analyzing data-plane programs focused on verification. Freire et al. [16] present an approach to verify security and correctness properties of P4 programs. Vera [52] finds common known bugs (e.g., invalid memory accesses) in P4 programs via symbolic execution. Similarly, p4pktgen [44] uses symbolic execution to automatically generate test cases for P4 programs and p4v [37] can verify P4 programs including their control-plane interface. While these approaches can be used with malicious intents to find exploitable bugs, they are limited in terms of which input they consider (typically only one packet). Recent work by Kanget al. [31] aims at finding attacks and defenses for P4 programs automatically. While [31] is limited to simple programs, we agree that symbolically executing P4 programs in order to find and fix vulnerabilities is a promising research direction (cf. §5).

Attacks against the forwarding behavior of a network Existing attacks demonstrate how to manipulate routing protocols. For example, to pollute forwarding tables (e.g., [9, 32, 56]) or to hijack traffic (e.g., [19, 38]). As we argue in this paper, the rise of programmable data planes greatly increases the attack surface.

Attacks against protocols There is a number of publications about attacking protocols such as IP (cf. [22]) and TCP (e.g., [20, 33, 48]) and—as we show in this paper—many novel protocols and algorithms allow new attacks.

7 CONCLUSION

Data-driven network control promises responsiveness and automation, and has great potential to overcome the shortcomings of traditional network control. However, the clear separation of the data- and the control channels has been a huge security advantage in terms of reducing the attack surface of networks. This is clearly not the case for data-driven networks acting on Internet traffic: *anybody* with an Internet connection can start injecting adversarial inputs into it.

In this work, we characterize the threat and highlight its destructive potential on multiple concrete use cases. We show that the impact is both real and worrying.

Not all is lost though. We do believe that there is a way to combine the benefits of data-driven decisions with the security benefits of data-agnostic ones. Doing so is challenging, but we highlight a promising research agenda in that direction.

To sum up, we hope that our work will encourage the community to start considering security as a first-class citizen when thinking about the next self-organizing, knowledge-defined, cognitive, or self-driving network technologies.

REFERENCES

[1] Platform Lab. 2019. https://platformlab.stanford.edu/platform-self-programming-networks.php.

[2] Aditya Akella, Bruce Maggs, Srinivasan Seshan, and Anees Shaikh. On the Performance Benefits of Multihoming Route Control. *IEEE/ACM ToN'18.*

[3] N. Akhtar and A. Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access'18.*

[4] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues. In *USENIX NSDI'20.*

[5] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *SOSP'01.*

[6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR'14.*

[7] Jake Brutlag. Speed Matters for Google Web Search. 2009. https://services.google.com/fh/files/blogs/google_delayexp.pdf.

[8] CAIDA. The CAIDA UCSD Anonymized 2013/2014/2015/2016/2018 Internet Traces. http://www.caida.org/data/passive/passive_2013_dataset.xml.

[9] CISCO. CAM Overflow - CCNP Security Secure 642-637 Quick Reference: Cisco Layer 2 Security. 2011. http://www.ciscopress.com/articles/article.asp?p=1681033&seqNum=2.

[10] David D Clark, Craig Partridge, J Christopher Ramming, and John T Wroclawski. A knowledge plane for the internet. In *ACM conference on Applications, technologies, architectures, and protocols for computer communications. 2003.*

[11] Igino Corona, Giorgio Giacinto, and Fabio Roli. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences, 2013.*

[12] Scott A. Crosby and Dan S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *USENIX Security'03.*

[13] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *USENIX NSDI'15.*

[14] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *IEEE Conference on Computer Vision and Pattern Recognition. 2018.*

[15] Nick Feamster and Jennifer Rexford. Why (and how) networks should run themselves. *arXiv, 2017.*

[16] Lucas Freire, Miguel Neves, Lucas Leal, Kirill Levchenko, Alberto Schaeffer-Filho, and Marinho Barcellos. Uncovering bugs in p4 programs with assertion-based verification. In *ACM SOSR'18.*

[17] T. Gerbet, A. Kumar, and C. Lauradoux. The Power of Evil Choices in Bloom Filters. In *IEEE/IFIP International Conference on Dependable Systems and Networks. 2015.*

[18] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. Dapper: Data Plane Performance Diagnosis of TCP. In *ACM SOSR'17.*

[19] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols. *ACM SIGCOMM CCR'11.*

[20] F. Gont. RFC 5927 - ICMP Attacks against TCP. 2010. https://tools.ietf.org/html/rfc5927.

[21] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security. 2017.*

[22] B. Harris and R. Hunt. TCP/IP security threats and attack methods. *Computer Communications. 1999.*

[23] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *USENIX NSDI'19.*

[24] Kuo-Feng Hsu, Ryan Beckett, Ang Chen, Jennifer Rexford, Praveen Tammana, and David Walker. Contra: A Programmable System for Performance-aware Routing. In *USENIX NSDI'20.*

[25] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv, 2017.*

[26] V. Jacobson. Congestion Avoidance and Control. *ACM SIGCOMM CCR'1988.*

[27] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. Data-Driven Networking: Harnessing the "Unreasonable Effectiveness of Data" in Network Design. 2016.

[28] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. Unleashing the potential of data-driven networking. In *International Conference on Communication Systems and Networks. 2017.*

[29] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *USENIX NSDI'17.*

[30] Daniel Kahneman. *Thinking, fast and slow. 2011.* Macmillan.

[31] Qiao Kang, Jiarong Xing, and Ang Chen. Automated attack discovery in data plane systems. In *USENIX Workshop on Cyber Security Experimentation and Test. 2019.*

[32] Alex Kirshon, Dima Gonikman, and Gabi Nakibly. Owning the Routing Table New OSPF Attacks. *BlackHat Briefings and Trainings USA+. 2011.*

[33] Aleksandar Kuzmanovic and Edward W Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *ACM conference on Applications, technologies, architectures, and protocols for computer communications. 2003.*

[34] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *HotNets'10.*

[35] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. LossRadar: Fast Detection of Lost Packets in Data Center Networks. In *ACM CoNEXT'16.*

[36] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. FlowRadar: A Better NetFlow for Data Centers. In *USENIX NSDI'16.*

[37] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. P4V: Practical Verification for Programmable Data Planes. In *ACM SIGCOMM'18.*

[38] Robert Lychev, Sharon Goldberg, and Michael Schapira. BGP Security in Partial Deployment: Is the Juice Worth the Squeeze?. In *ACM SIGCOMM'13.*

[39] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM'17.*

[40] Roland Meier, Petar Tsankov, Vincent Lenders, Laurent Vanbever, and Martin Vechev. NetHide: Secure and Practical Network Topology Obfuscation. In *USENIX Security'18.*

[41] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. Knowledge-defined networking. In *ACM SIGCOMM CCR'17.*

[42] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM'17.*

[43] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. Hardware-accelerated network control planes. In *HotNets'18.*

[44] Andres Nötzli, Jehandad Khan, Andy Fingerhut, Clark Barrett, and Peter Athanas. P4Pktgen: Automated Test Case Generation for P4 Programs. In *ACM SOSR'18.*

[45] Fabien A. P. Petitcolas. *Kerckhoffs' Principle.* Springer US. 2011.

[46] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the Network Be the AI Accelerator?. In *In ACM Morning Workshop on In-Network Computing. 2018.*

[47] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-Network Computation is a Dumb Idea Whose Time Has Come. In *HotNets'17.*

[48] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *ACM SIGCOMM CCR'99*.

[49] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *ACM SIGCOMM'17*.

[50] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM CCS'16*.

[51] Giuseppe Siracusano and Roberto Bifulco. In-network Neural Networks. *arXiv, 2018.*

[52] Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Debugging P4 Programs with Vera. In *ACM SIGCOMM'18*.

[53] Muhammad Usama, Junaid Qadir, and Ala Al-Fuqaha. Adversarial Attacks on Cognitive Self-Organizing Networks: The Challenge and the Way Forward. In *IEEE Conference on Local Computer Networks Workshops. 2018.*

[54] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to Route. In *HotNets'17*.

[55] Yaron Velner, Kalev Alpernas, Aurojit Panda, Alexander Rabinovich, Mooly Sagiv, Scott Shenker, and Sharon Shoham. Some complexity results for stateful network verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems. 2016.*

[56] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *ACM SIGCOMM'15*.

[57] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *NDSS'16*.

[58] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM'18*.

[59] H. Yao, C. Qiu, C. Fang, X. Chen, and F. R. Yu. A Novel Framework of Data-Driven Networking. *IEEE Access'16*.

[60] KK Yap et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *ACM SIGCOMM'17*.