NetCo: Reliable Routing With Unreliable Routers

Anja Feldmann[1], Philipp Heyder[1], Michael Kreutzer[2], Stefan Schmid[1,3]
Jean-Pierre Seifert[1,4], Haya Shulman[2], Kashyap Thimmaraju[1,4], Michael Waidner[2], Jens Sieberg[5]
[1] TU Berlin, Germany [2] Fraunhofer SIT, Germany [3] Aalborg University, Denmark [4] T-Labs Berlin, Germany [5] BSI, Germany

***Abstract*—Software-Defined Networks (SDNs) are typically designed and operated under the assumption that the underlying routers (and switches) are trustworthy. Recent incidents, however, suggest that this assumption is questionable. The possibility of incorrect or even malicious router behavior introduces a wide range of security problems. The problem is exacerbated by the fact that governments and companies do not have the expertise nor budget to build their own trusted high-performance routing hardware.**

This paper presents NetCo, an approach to build secure routing using insecure routers. NetCo is inspired by the robust combiner concept known from cryptography, and leverages redundancy to compile a secure whole from insecure parts. We present the basic design of NetCo, and report on a small prototype implementation in OpenFlow. We also sketch a virtualized version of NetCo which, by leveraging SDN traffic engineering flexibilities, can significantly reduce the hardware costs involved in implementing NetCo.

I. Introduction

Modern computer networks have become a critical infrastructure. Enterprise and datacenter networks as well as the Internet carry an increasing amount of critical and confidential traffic: from private users, companies, but also governments. This introduces strict requirements in terms of availability and security.

Software-Defined Networks (SDN) introduce interesting opportunities for designing more dependable computer networks, which meet these new requirements. In a nutshell, in an SDN, the control over the routers (resp. switches) is outsourced and consolidated to a logically centralized software, called the *controller*. The separation of control plane and data plane not only promises faster innovations, but also allows to radically simplify the required functionality in the data plane. This facilitates a formal reasoning about the function provided by the network and its correctness: a crucial prerquisite of any reliable network. For example, OpenFlow, the de facto SDN standard today, is based on a match action paradigm: the logic of the switch is defined by a set of match-action rules, where each match packet is processed according to the corresponding action.

While software-defined networks and in particular OpenFlow introduce a more verifiable and hence secure network, the paradigm critically depends on the correctness of the underlying hardware. However, the assumption of reliable routers and switches is questionable. Over the last years, attackers have repeatedly demonstrated their ability to compromise switches and routers [1], [2], [3], thousands of compromised access and core routers are being traded underground [4], networking vendors have left backdoors open [5], [6], national security agencies can bug network equipment [7], hacker tools to scan and eventually exploit routers with weak passwords, default settings are openly available on the Web, etc. As SDN is moving into production networks, it is naive to assume that similar issues will not arise for OpenFlow switches.

The problem is a fundamental one: even large enterprises or national security agencies often cannot afford and do not have the expertise to develop their own trusted, high-performance network hardware. Rather, they need to rely on untrusted hardware from untrusted vendors.

An unreliable routing system introduces several threats: for instance, wrongfully forwarded packets, i.e., packets following incorrect routes, may bypass security-critical components such as firewalls or intrusion detection/prevention systems, and may be able to enter or leave security critical zones. Forwarding or mirroring packets wrongfully can also be used to violate isolation requirements in multi-tenant datacenters, and generally, to exfiltrate sensitive information [8]. Moreover, a malicious router may also simply seek to overload the network, using a Denial-of-Service (DoS) attack.

While encryption may be used to mitigate some of these problems, cryptographic approaches require an additional infrastructure, and also come with overheads at runtime. This is undesirable especially in high-performance networks. Moreover, even encrypted traffic may leak sensitive information, e.g., about the times and frequency of communications. Finally, cryptographic solutions cannot deal with the DoS threat.

A. Our Contributions

Today, we lack tools to verify and enforce correct *router* (or, equivalently in this paper: *switch*) behavior. This paper is motivated by the question whether Software-Defined Networks (SDNs) providing reliable routing can be built even in scenarios where individual components (routers and switches) are untrusted.

Our approach relies on the observation that even in completely untrusted environments, it may be reasonable to rely on some minimal *non-cooperation assumptions*: routers from different vendors, or routers manifactored in different countries, may be unlikely to cooperate and exhibit the same misbehavior. Accordingly, the natural idea explored in this paper is whether inherent router diversity can be leveraged to detect or even prevent attacks.

In particular, we present the NetCo (robust network combiner) system, which is inspired by the *robust combiner* concept [9], [10] known from cryptography, and which applies this concept to networks. While our approach is applicable more generally, we in this paper focus on a software-defined networks.

We first introduce a basic version of NetCo and discuss different architectual variants. We report on a proof-of-concept implementation in C (and Mininet) which demonstrates the feasibility of our approach. We show that NetCo can scale to relatively high throughputs both for UDP and TCP traffic, and can constrain the effects of malicious behavior. Subsequently, we initiate the discussion of an interesting virtualized version of NetCo. This virtualized version comes with significantly lower hardware requirements and leverages the inherent traffic-engineering flexibilities introduced by the software-defined networking paradigm.

B. Background

The software-defined networking paradigm introduces innovative and flexible ways to define routing paths in the network. At the heart of a software-defined network operating system lies a control software, running on a set of servers. These controllers receive information and statistics from routers, and depending on this information as well as the policies they seek to implement, issue instructions to the routers/switches. OpenFlow follows a match-action paradigm: The controllers install (flow) rules on the routers which consist of a match and an action part; the packets (i.e., flows) matching a rule are subject to the corresponding action. That is, each router stores a set of tables which are managed by the controllers, and each table consists of a set of flow entries which specify expressions that need to be matched against the packet headers, as well as actions that are applied to the packet when a given expression is satisfied. Possible actions include dropping the packet, sending it to a given egress port, or modifying it, e.g., adding a tag. The match-action paradigm is attractive as it simplifies formal reasoning and enables policy verification.

By default, if a packet arrives at a router and does not match an existing rule, the packet (usually without payload if the router supports packet buffering) is forwarded to the controller. This event is called a *Packet-in*. Upon a *Packet-in* event, the controller can decide how to react to packets of the corresponding type, and add/delete/modify flow rules accordingly issuing *Flow-mod* messages to the router (and maybe to other routers proactively on this occasion as well). A controller can also decide to send out a packet explicitly from a router, issuing a so-called *Packet-out* command to the router.

We remark that throughout this paper, we will use the terms SDN and OpenFlow interchangeably, and also emphasize that there exist different and not always backward compatible OpenFlow versions. Our prototype is based on the OpenFlow 1.0 standard.

C. Organization

The remainder of this paper is organized as follows. Section II presents our threat model. Section III presents the main concepts underlying NetCo, Section IV describes an early prototype implementation, Section V reports on a performance evaluation, and Section VI discusses a case study: a routing attack in the datacenter. Section VII initiates the discussion of a more cost-effective version of NetCo based on SDN traffic engineering flexibilities. After reviewing related literature in Section VIII, we conclude our work in Section IX.

II. Threat Model

We consider a network consisting of a set of *routers* (for the purpose of this paper, usually: OpenFlow switches) R, connected by a set of *links* E, and managed by a logically centralized SDN controller.. We assume a scenario where a network provider or user (e.g., a government) cannot trust the routing hardware. In particular, designing and in-house fabrication of own trusted routers is impossible (e.g., due to the excessively high costs entailed by such a solution). Accordingly, we consider a strongly adversarial model where routers (resp. router vendors) can behave *arbitrarily*, e.g., completely ignore the installed OpenFlow match-action rules installed by the controller: That is, we do not place any restrictions on what an adversarial router can and cannot do. For example, a adversarial router can fabricate and transmit any type of message. In particular, routers may contain (hardware and software) backdoors.

For example, in an environment where traffic is not necessarily encrypted or integrity-protected, e.g., in a high-performance environment, a malicious router can manipulate packets in many ways, e.g., by changing VLAN identifiers, isolation properties can be violated. But also in a scenario where traffic is encrypted and protected, e.g., in a transport network, a malicious router may still attack the availability of the network: by dropping packets or duplicating packets, a Denial-of-Service (DoS) attack can be performed. See Figure 1 for an illustration of two example scenarios.

In general, an adversarial router may perform the following attacks.

1) **Rerouting:** An adversarial router can forward a packet to the wrong port (e.g., breaking logical isolations).
2) **Mirroring:** An adversarial router can duplicate a packet, and e.g., send one to the correct and one to an incorrect port.
3) **Packet Modification:** An adversarial router can also delete packets, generate new packets, or modify the header or payload of packets (e.g., changing the VLAN field to break isolation domains).
4) **Denial-of-Service (DoS):** An adversarial router may also generate a very large number of packets in order to overload the network: both the link resources as well as the (CPU or TCAM) resources of nearby network elements. A DoS attack can also be performed by dropping packets.

Moreover, we do not only make no assumptions about the behavior of malicious routers, but also not on the *number* of malicious routers. In principle, *all* routers may misbehave.

However, we assume bounds on the degree of malicious router collusion. In particular, we assume a certain degree of heterogeneity or diversity: routers fall into different categories,

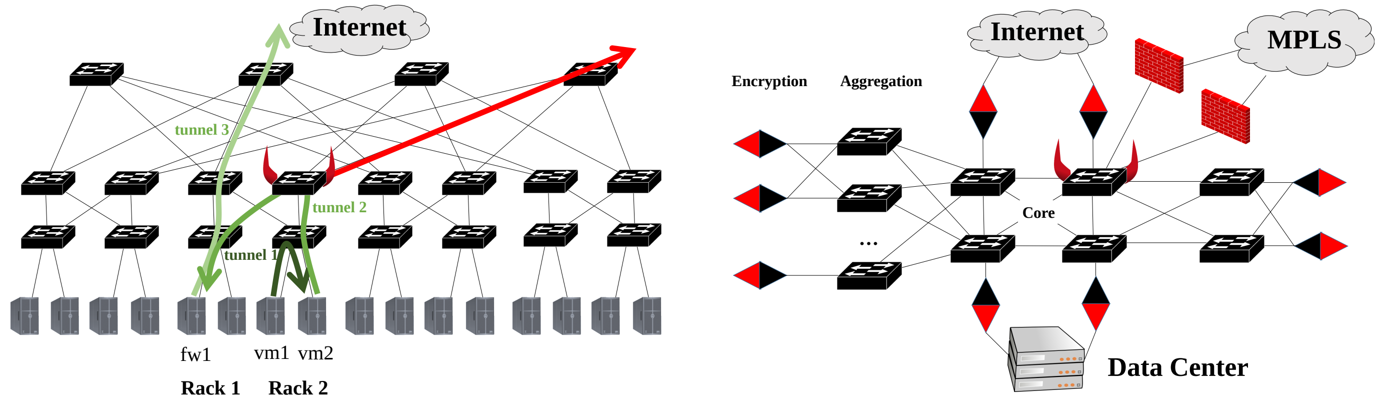

Fig. 1. Illustration of two possible scenarios and attacks on the routing protocol. *Left: High-Performance Datacenter Scenario.* The figure shows a typical fat-tree topology where servers are organized in racks, which are in turn organized in pods, interconnected by core routers. In this scenario, *rack 1* contains security-critical equipment such as firewalls, and *rack 2* contains different virtual machines (e.g., implementing a web service). A correct communication channel is indicated in *green* (using three tunnels): a virtual machine $vm1$ communicates to $vm2$, which in turn communicates to a client in the Internet via the firewall $fw1$ located in rack 1. However, a malicious aggregation router (depicted with *red horns*) may aim to manipulate unprotected packet headers such that confidential information can be exfiltrated by bypassing the firewall (route in *red*). *Right: Crypto Transport Scenario.* In this scenario, a transport network is shown where all traffic is encrypted at the edge (indicated by *red-black diamonds*). Due to the cryptographic protection, an attacker (*red horns*) cannot easily manipulate the correctness of routing. However, it can target the *availability* of the network, e.g., by launching a Denial-of-Service attack.

and are unlikely to cooperate across category boundaries (the adversarial strategy is not synchronized). For example, a natural category could be the *vendor type*: switches and routers from different vendors may be unlikely to exhibit the same bugs or misbehaviors. In the context of hardware backdoors, a reasonable category may concern the *locations and countries* where the hardware were actually fabricated.

Moreover, while we assume that developing own trusted high-performance routing hardware is too costly, we argue that designing simple trusted components can be feasible. In other words, trusted components can be introduced as long as their functionality is simple enough to facilitate a cheap production. This is a reasonable and interesting assumption: if a government can increase the availability and security of the network by introducing a small number of cheap trusted hardware boxes, it is likely to do so.

III. The NetCo Approach

Can we build a reliable routing system based on untrusted routing hardware, even if *all* routers are adversarial? Perhaps surprisingly at first sight, the answer is yes, and the approach proposed in this paper is based on two key concepts:

1) *Leverage redundancy and diversity:* Given a certain hardware heterogeneity which renders collusion among different routers unlikely, we can assemble and connect these routers in a manner which allows us to detect and prevent misbehaviors.
2) *Trusted but simple components:* While high-peformance routers are complex and an in-house production often out-of-question, the design of simpler and trusted components may be feasible at low costs.

Our approach is inspired by the robust combiner concept known from cryptography [9]. Concretely, we show that with two simple trusted components, a so-called *hub* and a so-called *compare*, a trusted router can be emulated based on untrusted routers.

The basic idea of our proposed NetCo approach is to apply the robust combiner approach to networks: we replace each router r in the network by a number k of different routers, say $k = 3$: $\{r_1, r_2, r_3\}$. These three routers are organized in a parallel circuit: the traffic originally input entering r is now forwarded by the *hub*, to each of the three routers r_i, $i \in \{1, 2, 3\}$. After the routers processed the packets and forwarded them to their outports, they reach the *compare*: the logic which lies at the heart of NetCo. In the most simple case, the *compare* can simply perform a *majority decision*: a packet is only forwarded if it is received by at least two routers (in general: by more than $\lfloor k/2 \rfloor$ routers). Moreover, depending on the threat model, packets may be compared bit-by-bit, or just based on the header, or hashing can be used.

Figure 2 provides a rough overview of the NetCo concept. In general, we observe that the number of required parallel routers depends on the required protection: for detecting misbehavior, two are enough, for prevention, we need three.

IV. An Early Prototype

While our approach can in principle be used to improve the security of any network based on switches and routers, in our prototype implementation, we so far focus on networks based on OpenFlow switches. In the following, we first describe the general design and then discuss our Mininet implementation (in C) in more details.

The implementation of the *hub*s is simple and can be realized in the datapath: the logic boils down to multiplying the packets, in a stateless manner. The *compare* logic however is more complex. For the sake of our prototype, we use a single, centralized server process for the *compare*. The *compare* is connected to the data plane akin of an OpenFlow controller, using *packet-in* and *packet-out* messages. However,

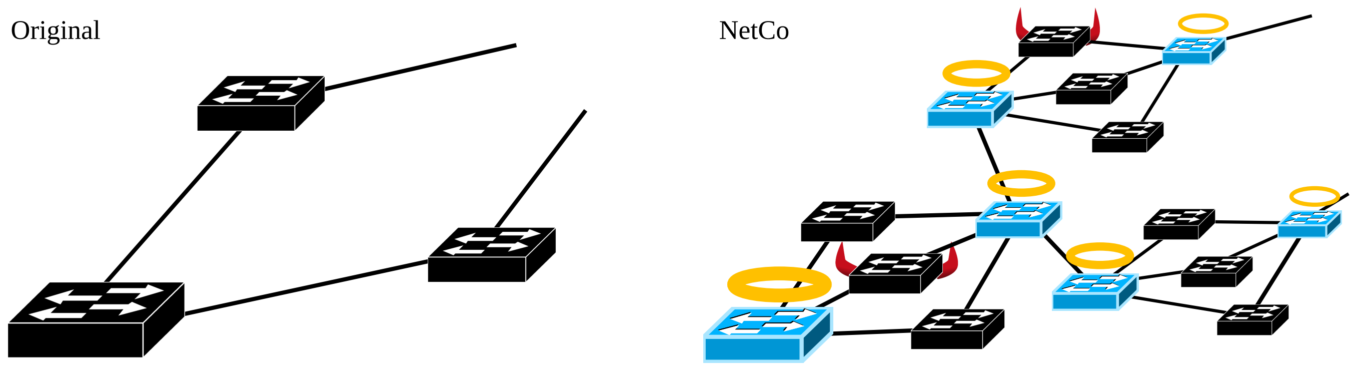

Fig. 2. NetCo Overview: On the *left*, an excerpt of the original network is shown. On the *right*, the corresponding robust combiner is shown: Each router is replaced by a *hub*, followed by three redundant routers in parallel, followed by a *compare*. The trusted components are colored *blue* (and with halo), the untrusted in *black*. The influence of a malicious router (with *red horns*) can be limited.

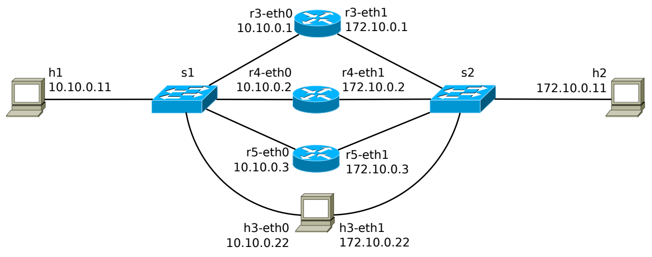

Fig. 3. Topology used for performance testing.

the *compare* is *not* running as an SDN application: obviously, and as we will see, this results in a poor performance. Rather, the *compare* is implemented directly and in C. Alternatively, the *compare* could also be implemented inband, e.g., as a middlebox, or in the context of Network Function Virtualization (NFV), as a virtualized network function.

Let us consider the example in Figure 3: it will also be our reference testing topology. In this $k = 3$ combiner architecture, $h1$ and $h2$ are benign hosts, and $h3$ is the (trusted) server running our combining module. The routers are untrusted components. Components $s1$ and $s2$ act either as *hub* or manage the traffic to and from the *compare*, depending on the direction in which a packet flows. In our prototype, they are implemented as OpenFlow switches as well, but their functionality can be much simpler, and hence realized as a trusted component.

The topology is created within the Mininet network emulation environment. Components $s1$ and $s2$ have a very simple set of flow rules: Every packet entering NetCo is forwarded to each r_i. Every packet received from any r_i is forwarded to the *compare*, after ensuring its ingress port number matches its MAC source address. Every packet received from the *compare* is to be forwarded based on the switches MAC table.

The *compare* program itself runs on a dedicated host (e.g., as a virtual network function). Each packet entering the *compare* is received by an Ethernet socket (an OpenFlow Packet-in) and saved in a structure containing the full packet as well as the ingress port number. In order to prevent resource attacks on this structure, the different buffers should be (logically) isolated. Using *memcmp()* the packet is then compared (bit-by-bit) to the packets already cached by the *compare*. If a match is found, the appropriate ingress port number of the cached element is incremented. Once a packet has been received on the majority of the possible ingress ports, the *compare* releases it *immediately*: A single copy of the packet is sent back to the switch (an OpenFlow Packet-out), which then forwards it according to the decision the majority of the r_i made; if additional packets (beyond the majority, e.g., if all switches are benign) arrive later, they are ignored.

In an ideal world, a packet arrives at each ingress port once. However, in the presence of malicious routers, we have to accommodate for the following cases as well:

1) **Packet received on one ingress port only:** This can happen, for example, if a malicious router rewrites the packet header field to send it on a path for exfiltration. We also see this behavior if a router starts crafting packets unsolicited. NetCo deals with this case by keeping the unique packet in the buffer for some time, and eventually deleting it (it is not sent out).
2) **Packet received on one ingress port multiple times:** This is the case, for example, if a router aims to launch a denial-of-service attack, striving to overwhelm a network component. As descriped in the above case, packets only arriving on one ingress port will not travel past the *compare*. Moreover, in the case of a denial-of-service attack, the *compare* may advice the corresponding switch to block the appropriate port for some time.
3) **Packets are not received on a particular ingress port:** If a number of consecutive packets were not received on a particular ingress port, the *compare* assumes that the router connected to that port is unavailable. This raises an alarm to the network administrator.

Note that our construction should bound the waiting time for the majority of the packets to arrive, otherwise it is exposed to denial-of-service attacks. The time a packet should be kept in the buffer is a function of the latencies of all the connected devices and the links.

We conclude by noting that our prototype implementation is based on OpenFlow 1.0. The only matched header field is the MAC destination address, and the only written header field is the MAC source address.

V. Performance Evaluation

Based on our prototype, we conducted a first study of the achievable performance of NetCo.

A. Evaluation Methodology

We evaluate the performance of our NetCo prototype through a series of different scenarios. In particular, we are interested in the tradeoff between performance and security. On one end of the spectrum, we consider a completely insecure network (called **Linespeed**) without robust combiner, which however also avoids the overheads introduced by the combiner. We then consider a weak, $k = 3$-parallel combiner (tolerating a single malicious switch) and a strong, $k = 5$-parallel combiner (tolerating two malicious switches); the former is called **Central3**, the latter **Central5**. For comparison, we compare the performance of our C-based *compare* to a *compare* implemented as a POX controller application **POX**. Finally, to study the impact of combining, we also consider reduced robust combiner designs where packets are only split, but not properly combined (**Dup3** for $k = 3$ and **Dup5** for $k = 5$). All scenarios are derived from Figure 3. In summary:

1) **Linespeed:** The simplest abstraction of our testing topology features only $h1$, $s1$, $r3$, $s2$ and $h2$. A benchmark for the ideal performance, which informs our expectations.
2) **Central3:** The full prototype implementation described in Figure 3, featuring $k = 3$ test routers.
3) **Central5:** The full prototype implementation described in Figure 3, featuring $k = 5$ test routers.
4) **POX3:** A reference implementation of NetCo as a SDN application running on the POX controller instead of $h3$ and testing three routers.
5) **Dup3:** Nodes $s1$ and $s2$ act as *hub*s, duplicate packets are not removed. Three test routers are put in a parallel circuit.
6) **Dup5:** Nodes $s1$ and $s2$ act as *hub*s, duplicate packets are not removed. Five test routers are put in a parallel circuit.

Our measurements are obtained using *iperf*. For the first 10 measurements, $h1$ acts as client and $h2$ as the server. These roles are reversed in the following 10 test runs. Each test run lasts 10 seconds. We usually present the average over 10 test runs for each scenario, when setting the *iperf -u* flag and adjusting the *-b* flag value until a maximum is reached.

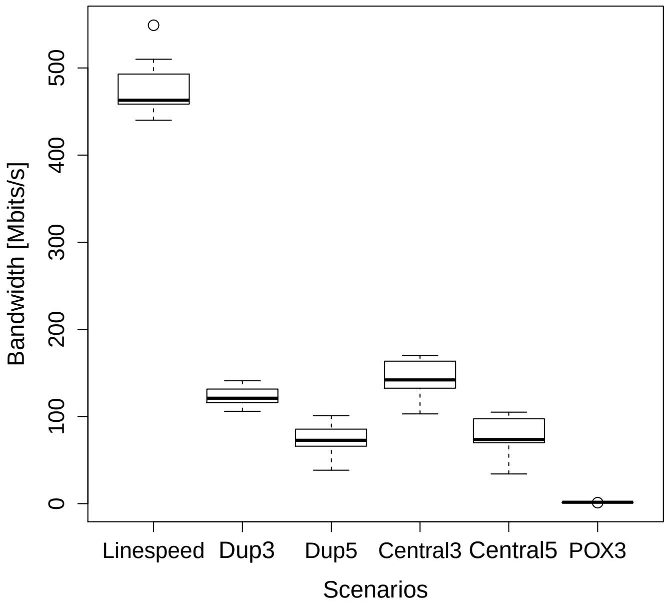

Fig. 4. TCP throughput

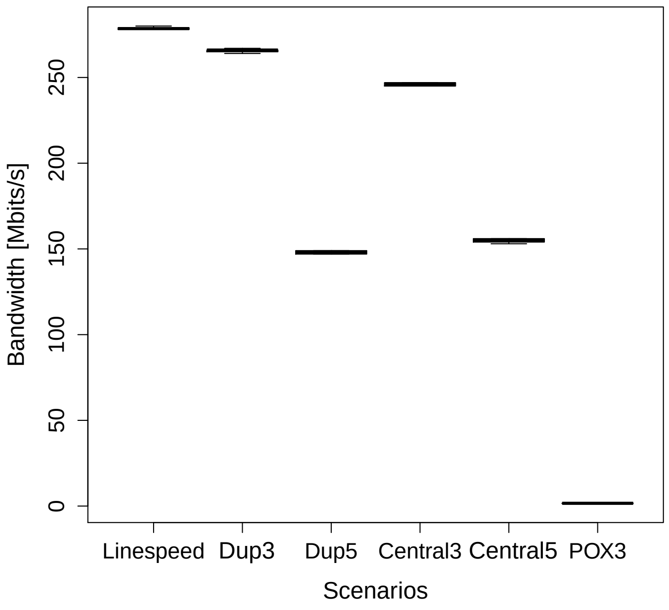

Fig. 5. UDP throughput

B. Results

Let us first give an overview of the plots and then discuss the observed results in detail. The TCP throughput for the six scenarios is shown in Figure 4. The maximum UDP throughput for loss rates below 0.5% is depicted in Figure 5. For our reference scenario *Central3*, we explore the connection between UDP throughput and loss rate. Figure 6 shows this relationship. Figure 7 shows the results of pings between $h1$ and $h2$. Each bar represents the average of three sequences of 50 consecutive ICMP request response cycles. Table I provides an overview of the averages measured for each

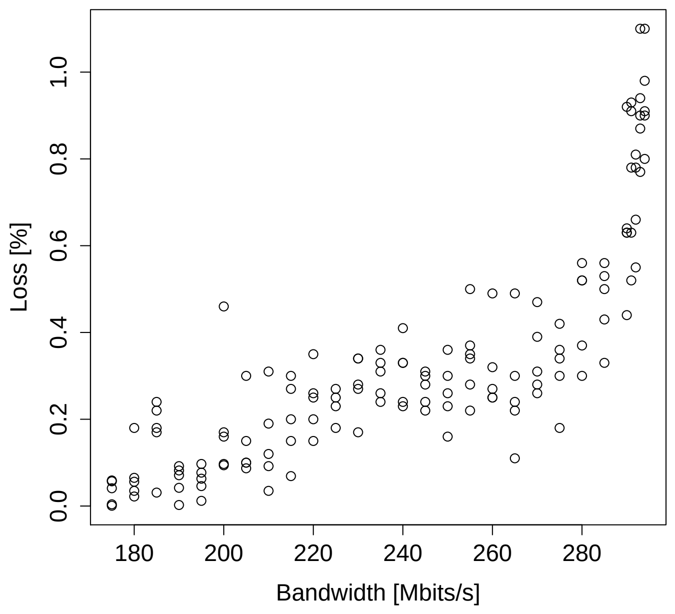

Fig. 6. Correlation of throughput and loss rate, in *Central3* scenario.

scenario. Finally, we also examine the jitter each scenario exhibits for varying UDP packet sizes. Figure 8 shows the corresponding results, each bar representing the average of five measurements.

TABLE I
AVERAGE MEASUREMENT RESULTS

	Linespeed	Dup3	Dup5	Central3	Central5
avg tcp bandwidth in Mbits/s	474	122	72	145	78
avg udp bandwidth in Mbits/s	278	266	149	245	156
avg RTT in ms	0.181	0.189	0.26	0.319	0.415

A first general observation we can make is that *security comes at a price:* Compared to the Linespeed scenario, the robust combiner scenarios show lower performance indicators. Both the Round-Trip-Time (RTT) increases and TCP as well as UDP throughput decreases, when moving from the Linespeed scenario to $k = 3$, and from $k = 3$ to $k = 5$. Therefore, the security benefits must be weighted against their performance implications. In the following, we offer some insights into this performance tradeoff.

Figure 4, 5 and 7 confirm our expectation, that the more untrusted routers NetCo contains, the lower the bandwidth and the higher the different RTT stats become: More packets are in flight in *Dup5* and *Central5* than in *Dup3* and *Central3*; thus packets spend more time buffered on exiting the NetCo design and the destination host. We however also observe that removing the duplicate packets (by combining) increases the throughput visibly. Combining also improves latency: the benefit we gain from lower buffering times at the destination

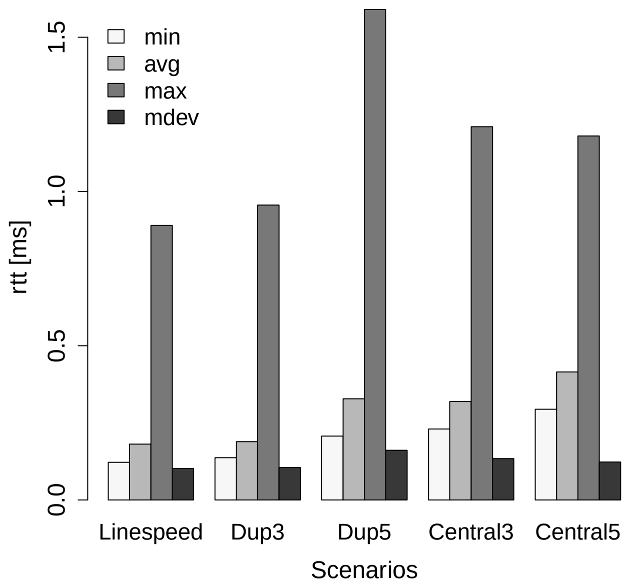

Fig. 7. Ping round-trip-time for five different scenarios.

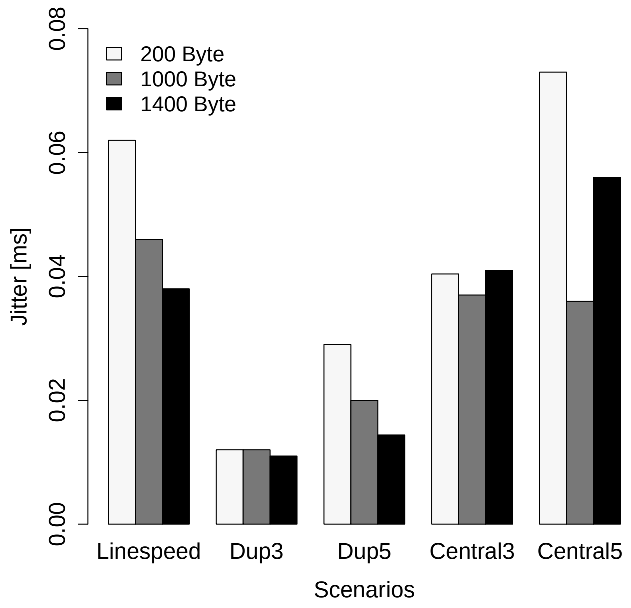

Fig. 8. Jitter for varying packet sizes.

host seems to outweigh the transmission delay added by the extra link and the processing delay added by the *compare*.

When comparing Figure 4 and Figure 5, we notice that the test scenarios better approximate the benchmark scenario Linespeed when packets are exchanged using connectionless UDP. We attribute this to TCP's congestion control mechanism: Packet loss triggers slow start, thus the more often packets are lost, the slower the connection becomes. In the UDP based tests we actively kept the loss rate low for all scenarios. This leads to a lower throughput in the Linespeed scenario, but to a higher throughput over all testing scenarios: the impact of a packet loss on the sending rate is not as severe.

We learn from Figure 8 that bigger packets lead to lower jitter. NetCo meets that expectation. A flow of many small packets fill up the packet cache of the *compare* more quickly than a flow of fewer, but larger packets. Once the packet cache is full, a clean up procedure starts, and we see that the more frequently the cache is cleaned up, the higher the jitter becomes.

Lastly the comparatively poor performance of the POX3 scenario can be explained by two key factors: (1) The choice of programming language: precompiled C code is generally executed much faster than interpreted Python (the language the POX controller is written in). (2) The controller delay, as piping every packet through the controller adds significant processing delay compared to a direct Ethernet socket on the wire.

VI. Case Study: Datacenter Routting Attack

As a case study for the robustness of NetCo, let us consider a specific attack. In particular, we revisit the threat model introduced in Section II, where a malicious router, located at an aggregation switch in a Clos datacenter, mirrors and drops packets.

In order to emulate such an attack, we set up the Mininet network with routing based on MAC destination addresses. We use ICMP echo requests and responses as test packets that traverse the path indicated as *tunnel 2*: An echo request sent from $vm1$ to $fw1$ travels over an edge switch, an aggregation switch and another edge switch, until reaching its destination. The response takes the same path in reverse. We call this path the *benign path*. Ten ICMP request response cycles were initiated in each of the following scenarios.

In a baseline scenario, all switches are benign. We observe at $vm1$ that all ICMP echo requests are received and answered correctly. Furthermore, we use two screening methods in parallel to ensure, that packets do not stray from the benign path: Using tcpdump to monitor packet arrivals on all interfaces adjacent to the benign path, as well as by monitoring the flow table counters of all switches. We verify that our test packets travelled the above mentioned benign path and no copies are received on any other node. Thus we witness 10 perfect cycles.

In a second scenario, to simulate malicious behaviour, we extend the flow rules in the following way: The aggregation switch now mirrors packets intended to reach $fw1$ to a core switch (following the red path in Figure reffig:scenarios). Also all packets matching $vm1$'s destination MAC address are to be dropped. After starting the ICMP requests in this scenario, we verify that the mirrored requests arrive at the core switch. From here, the copies are forwarded to $fw1$. Firewall $fw1$ then sends two responses (one to the benign request and one for the mirrored copy), both of which are dropped at the aggregation switch. We did not record any other stray packets.

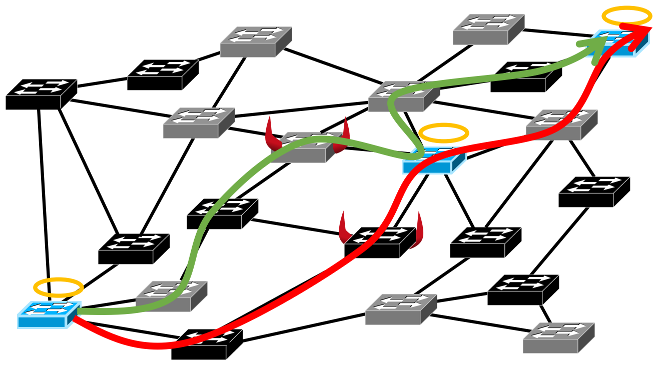

Fig. 9. Illustration of the virtualized NetCo. The robust combiner is emulated only: leveraging SDN traffic engineering flexibilities or tunneling to steer flows through a heterogeneous set of devices (e.g., one vendor in *black*, the other one in *grey*), the *compare* is implemented inband.

To summarize: After 10 requests sent, we witness 20 requests arriving at $fw1$ and 0 responses arriving at $vm1$.

In a third scenario, the malicious aggregation switch is placed in NetCo along with two benign switches. Repeating the ICMP echo test, we did not see any packet stray from the benign path. The most interesting node to monitor, however was the server running the *compare*. Indeed, we saw the mirrored packets arriving, yet none of them left the *compare*: As all of them were received from only one of the three candidates, they could never win the majority decision and were subsequently dropped. We also noticed, that only two copies of each response reached the *compare*. However since two out of three constitutes a majority, one copy of each response was released by the *compare* and forwarded to $vm1$. Thus all 10 request response cycles completed successfully.

VII. The Virtualized NetCo

A main disadvantage of the NetCo approach described so far is the need for physical redundancy, which can entail costly infrastructure investments. However, we argue that such investments can sometimes be avoided, by virtualizing the NetCo concept. While the fundamental idea underlying NetCo is to use redundancy to detect and prevent attacks, we observe that such redundancy is often readily available or can be introduced using the natural life cycles of the network components.

Virtualizing here means that the redundant links used in the NetCo architecture do not necessarily have to be physical ones, but could be virtual and constitute *paths*: splitting a flow into two (for detection) or three (for prevention) copies along different segments of the path, using *tunneling*, has a similar effect as in the physical robust combiner approach: the resulting hardware diversity allows us to make majority decisions and filter out undesired behavior. Figure 9 illustrates the idea of the virtualized NetCo.

We believe that Software-Defined Networks (SDNs) provide an ideal environment to implement such virtualized combiners. due to their traffic engineering flexibilities, i.e., in terms of how flows can be defined and steered through the network: the match-action rules on an OpenFlow switch enable a forwarding and routing logic which can depend on layer-2, layer-3 and layer-4 header fields (and sometimes beyond), and which is not limited to shortest paths.

VIII. Related Work

Interestingly, while over the last years, much research was conducted on how to secure routing protocols on the control plane [11], [12], [13], [14], [15], providing authenticity and correctness of topology propagation and route computation, the important question of how to secure the data plane has received much less attention so far. In fact, until very recently, researchers did not even know whether it was possible to build a secure path verification mechanism [16]. Many existing systems like VeriFlow [17], Anteater [18] and Header Space Analysis [19] rely either on flow rules installed at routers or on data plane configuration information to perform their analysis. This information can easily be manipulated in malicious settings. While first interesting and more specific solutions like ICING [16] or Stealth Probing [20] as well as first secure variants of classic tools such as traceroute [21] are emerging, providing fundamental properties like path consent and path compliance [16], but they are usually based on expensive cryptographic techniques.

SDNs are known to introduce many flexibilities, also in terms of security. For instance, Yu et al. [22] presented a distributed traffic monitoring scheme for SDNs, and FleXam [23] is a sampling extension for monitoring and security applications in OpenFlow. NetSight [24] leverages SDN to trace entire packet histories (without sampling), by collecting them "out-of-band", and CherryPick [25] uses packets to carry information of SDN paths "inband" (namely of a subset of links along the packet trajectory), however, these protocols are not robust to malicious routers. Bates et al. [26] use SDN networks (plus some middleboxes) to observe the data plane behavior, even in the presence of malicious routers. The traffic engineering flexibilities of SDN have also been exploited to perform secret sharing [27].

Our approach is inspired by the concept of robust combiners known from crypto literature [9], [10]. However, except for a recent parallel work by Achenbach et al. [28] in the context of firewalls, we are not aware of any applications of the robust combiner in the network domain.

IX. Conclusion

This paper is motivated by the observation that networks today on the one hand constitute a critical infrastructure, and on the other hand are vulnerable to an increasing number of attacks, including hardware backdoors in the routers. We have have argued that by leveraging diversity, in terms of vendors or countries where routers have been manufactured, an opportunity arises to deal with this seemingly inherent problem.

We understand our work as a first step, and believe that our model opens an interesting research area. In particular, the prototype we described in this paper obviously represents a very early stage of NetCo, and more extensive evaluations are necessary to optimize future implementations in terms of jitter, throughput and RTT characteristics. We also need to explore alternative architectures, which, e.g., implement the compare function inband, as a middlebox or NFV function. Moreover, we note that for detection and depending on the threat vector, the *compare* element does not necessarily have to be located in the data plane. An efficient alternative could be to reduce load on the *compare* using *sampling*: a simple logic in the data plane forwards a random subset of packets to a more thorough out-of-band *compare* logic. We also need to explore further the minimal requirements on the *hub* and *compare* functionality, such that true performance isolation properties are guaranteed under different attack patterns. Also, while we have so far focused on building a secure router out of insecure OpenFlow switches, we believe that our approach can easily be extended to legacy routers.

More generally, we believe that the main principle underlying NetCo, namely that of leveraging heterogeneity, is of broad interest and can find applications in many other domains. For example, for cost and performance reasons, our robust combiner approach could be used only in specific parts of the network where security is particularly critical or where other approaches fail (e.g., at the edge of the datacenter). The robust combiner concept could also implemented on a more coarse-granular level: for instance, a security critical transport network could be duplicated entirely, splitting and combining traffic only at the ingress and outgress, respectively. Moreover, we in this paper have started investigating the possibility to implement a robust combiner *logically*, without the need to invest into additional hardware. The opportunities and limitations of such alternative architectures need to be explored in future research.

We will release the code of our NetCo prototype to the community together with this paper.

Acknowledgments

Research supported by the German Federal Office for Information Security (BSI).

References

[1] "The tale of one thousand and one dsl modems," https://securelist.com/analysis/publications/57776/the-tale-of-one-thousand-and-one-dsl-modems/, 2012.

[2] "Synful knock - a cisco router implant - part i," https://www.fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html, 2015.

[3] F. Lindner, "Cisco ios router exploitation," *Black Hat USA*, 2009.

[4] S. Lee, T. Wong, and H. S. Kim, "Secure split assignment trajectory sampling: A malicious router detection system," in *Proc. International Conference on Dependable Systems and Networks (DSN)*, 2006, pp. 333–342.

[5] "Huawei hg8245 backdoor and remote access," http://websec.ca/advisories/view/Huawei-web-backdoor-and-remote-access, 2013.

[6] "Netis routers leave wide open backdoor," http://blog.trendmicro.com/trendlabs-security-intelligence/netis-routers-leave-wide-open-backdoor/, 2014.

[7] "Snowden: The NSA planted backdoors in cisco products," http://www.infoworld.com/article/2608141/internet-privacy/snowden--the-nsa-planted\\-backdoors-in-cisco-products.html, 2014.

[8] G. Kurtz, “Operation aurora hit google,” in *http://securityinnovator: com/index.php?articleID=42948&sectionID=25*, 2010.

[9] A. Herzberg, “Tolerant combiners: Resilient cryptographic design,” Cryptology ePrint Archive, Report 2002/135, Tech. Rep., 2002.

[10] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen, “On robust combiners for oblivious transfer and other primitives,” in *Advances in Cryptology (EUROCRYPT)*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., 2005, vol. 3494.

[11] W. Aiello, J. Ioannidis, and P. McDaniel, “Origin authentication in interdomain routing,” in *Proc. 10th ACM Conference on Computer and Communications Security (CCS)*, 2003, pp. 165–178.

[12] K. Butler, T. Farley, P. McDaniel, and J. Rexford, “A survey of bgp security issues and solutions,” *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[13] Y.-C. Hu and A. Perrig, “A survey of secure wireless ad hoc routing,” *IEEE Security and Privacy*, vol. 2, no. 3, pp. 28–39, May 2004.

[14] S. Kent, C. Lynn, and K. Seo, “Secure border gateway protocol (s-bgp),” *IEEE J.Sel. A. Commun.*, vol. 18, no. 4, pp. 582–592, Sep. 2006.

[15] L. Subramanian, “Decentralized security mechanisms for routing protocols,” Ph.D. dissertation, 2005.

[16] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, “Verifying and enforcing network paths with icing,” in *Proc. ACM CoNEXT*, 2011, pp. 30:1–30:12.

[17] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, “Veriflow: Verifying network-wide invariants in real time,” in *Proc. 10th USENIX NSDI*, 2013.

[18] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, “Debugging the data plane with anteater,” in *Proc. ACM SIGCOMM*, 2011, pp. 290–301.

[19] P. Kazemian, G. Varghese, and N. McKeown, “Header space analysis: Static checking for networks,” in *Proc. 9th USENIX NSDI*, 2012.

[20] I. Avramopoulos and J. Rexford, “Stealth probing: Efficient data-plane security for ip routing,” in *Proc. USENIX Annual Technical Conference (ATEC)*, 2006.

[21] G. Mathur, V. N. Padmanabhan, and D. R. Simon, “Securing routing in open networks using secure traceroute,” Microsoft Research, Tech. Rep. MSR-TR-2004-66, July 2004.

[22] Y. Yu, C. Qian, and X. Li, “Distributed and collaborative traffic monitoring in software defined networks,” in *Proc. ACM HotSDN*, 2014, pp. 85–90.

[23] S. Shirali-Shahreza and Y. Ganjali, “Flexam: Flexible sampling extension for monitoring and security applications in openflow,” in *Proc. ACM HotSDN*, 2013, pp. 167–168.

[24] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, “I know what your packet did last hop: Using packet histories to troubleshoot networks,” in *Proc. 11th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 71–85.

[25] P. Tammana, R. Agarwal, and M. Lee, “Cherrypick: Tracing packet trajectory in software-defined datacenter networks,” in *Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 2015, pp. 23:1–23:7.

[26] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, “Let SDN be your eyes: Secure forensics in data center networks,” in *Proc. NDSS Workshop on Security of Emerging Network Technologies (SENT’14)*, Feb. 2014.

[27] S. Dolev and S. Tzur-David, “Sdn-based private interconnection,” in *Proc. ACM PODC Workshop on Distributed Software-Defined Networks (DSDN)*, 2014.

[28] D. Achenbach, J. Müller-Quade, and J. Rill, *Proc. BalkanCryptSec*, 2015, ch. Universally Composable Firewall Architectures Using Trusted Hardware, pp. 57–74.