BlockSDN: Blockchain as a Service for Software Defined Networking in Smart City Applications

Gagangeet Singh Aujla, Maninderpal Singh, Arnab Bose, Neeraj Kumar, Guangjie Han, and Rajkumar Buyya

Abstract

Smart cities have emerged as a hub of intelligent applications (e.g., intelligent transportation systems, smart parking, smart homes, and e-healthcare) to provide ambient-assisted living and quality of experience to wide communities of users. The smooth execution of these applications depends on reliable data transmission between various smart devices and machines. However, the exponential increase in data traffic due to the growing dependency of end users on smart city applications has created various bottlenecks (e.g., channel congestion, manual flow configurations, limited scalability, and low flexibility) on the conventional network backbone, which can degrade the performance of any designed solution in this environment. To mitigate these challenges, SDN emerges as a powerful new technology that provides global visibility of the network by decoupling the control logic from the forwarding devices. The abstraction of network services in SDN architecture provides more flexibility for network administrators to execute various applications. In SDN architecture, the decision making process is handled by a logically centralized controller, which may have a single point of failure. An adversary/attacker can compromise the controller using different types of attacks (e.g., eavesdropping, man-in-the middle attack, and distributed denial of service) in order to gain total control of the network by updating the flow table entries at the data plane or hindering control plane operations. Therefore, to cope with the aforementioned challenges, new strategies and solutions are required for securing the SDN-enabled network architecture at different planes and their associated interconnections. In this article, various security issues and different attack vectors are discussed along with possible solutions. To mitigate various attacks, BlockSDN, a blockchain as a service framework, for SDN is proposed. The architecture of permissioned blockchain is presented followed by two attack scenarios, 1) a malware compromised switch at the data plane and 2) distributed denial of service attack at the control plane, to demonstrate the applicability of the BlockSDN framework for various future applications. Finally, the open issues and challenges with respect to the design of blockchain solutions for SDN in smart city applications are also discussed.

Introduction

The growth of smart cities has unfolded a diverse range of intelligent and ubiquitous applications ranging from intelligent transportation systems (e.g., smart parking and intelligent traffic lighting) to ambient living (e.g., smart homes, utilities, and services) to e-healthcare (e.g., smart hospitals and recommendation systems), to name a few. The evolution of the latest technologies such as the Internet of Things (IoT), cloud computing, and edge computing have provided a strong platform for the growth of the various aforementioned smart city applications [1]. Figure 1 shows different types of smart city applications and the enabling technologies. These data-intensive applications have emerged as a source of big data generation in a smart city environment. For example, a typical smart city housing a population of 1 million is expected to generate 200 million GB data per day [2]. Now, this data has to be effectively collected, processed, and analyzed for the continuous and non-disruptive provisioning of smart city applications. Thus, a massive amount of data needs to be transmitted at a rapid pace over the underlying forwarding devices. Therefore, relaying such a huge amount of data may lead to congestion on the underlying network backbone infrastructure, which in most cases is managed by TCP/IP. Although there are various congestion control algorithms proposed in the literature, these algorithms do not perform well in view of the data generated from different devices. Moreover, the dependence of conventional networks on standard protocols and algorithms acts as a potential bottleneck for handling the dynamic requirements of data transmission in a smart city environment [3].

Smart cities intend not only to provide better quality of experience to users but also to ensure robust and reliable connectivity for ubiquitous service provisioning. This requires improved bandwidth capabilities, enhanced flexibility, and a scalable service-oriented architecture. In this context, software defined networking (SDN) has come up as a critical network architecture for bearing the escalated challenges of providing flexible and scalable service-oriented architecture for smart city applications [3]. SDN provides various benefits due to its inherent architecture wherein the control capabilities are shifted to a centralized

Gagangeet Singh Aujla is with Newcastle University and Chandigarh University; Maninderpal Singh and Arnab Bose are with Chandigarh University; Neeraj Kumar is with Thapar Institute of Engineering and Technology; Guangjie Han is with Nanjing Agricultural University and Hohai University; Rajkumar Buyya is with the University of Melbourne.

Digital Object Identifier:
10.1109/MNET.001.1900151

0890-8044/20/$25.00 © 2020 IEEE

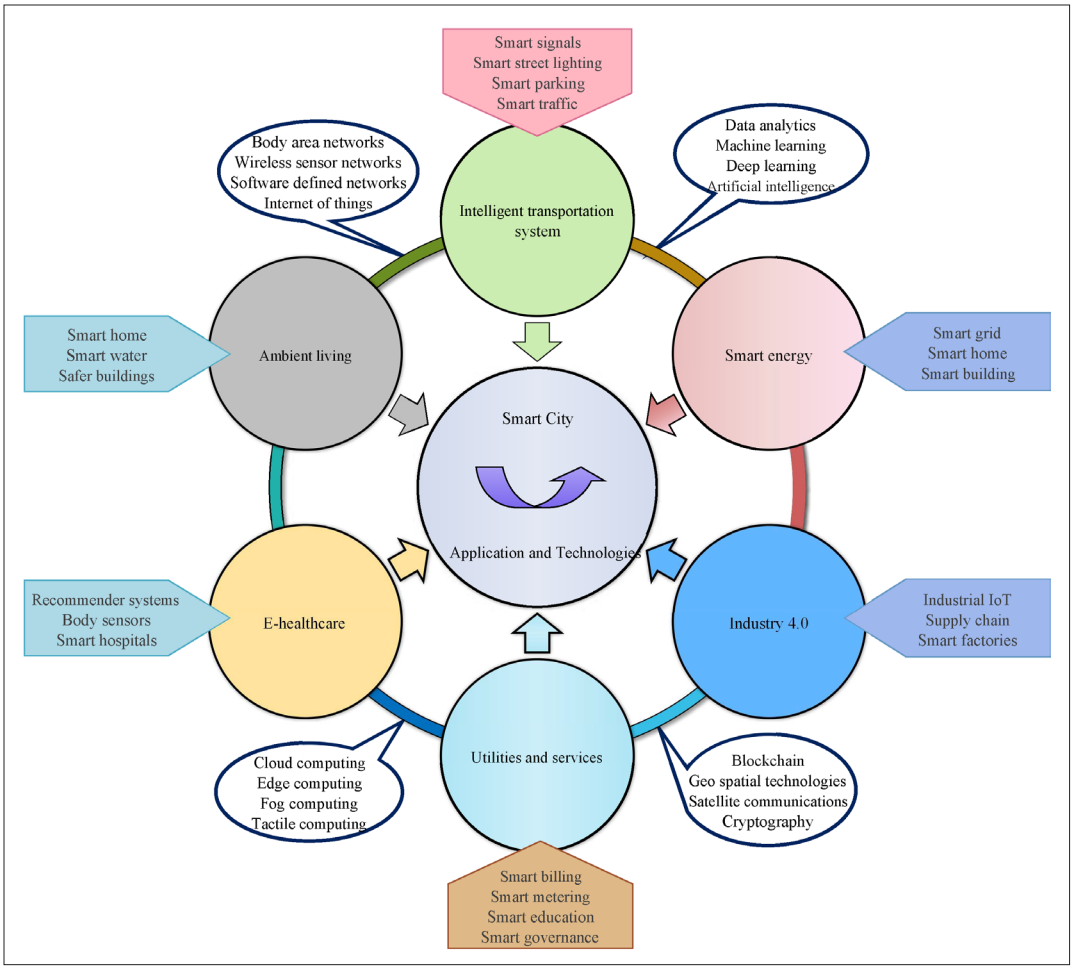

FIGURE 1. Smart city: applications and technologies.

controller rather than embedded on the forwarding devices. It offers increased coverage and capacity in a cost-effective manner by providing global network visibility. For example, multiple sites across the city may be connected and controlled from a centralized location, which in turn reduces the cost and complexity of deploying geo-distributed network resources. Moreover, the centralized control and visibility help to determine the best route according to the application flow requirement. It also helps to track the link health, congestion level, and application priority. SDN also allows the integration of services and cloud applications, which can run using a virtualized environment. It also provides the ability to provide multiple routes for a specific application, thereby providing a better customer experience when an application is accessed by multiple tenants. The key to the success of SDN is vendor neutrality and the use of open standards. Smart city applications can adopt multi-vendor architecture to reduce costs by pooling the computing, processing, and storage functionality [4].

Besides providing manifold benefits, SDN architecture is susceptible to different attacks and security concerns. The SDN architecture acts as a double-edged sword, which provides various benefits we well as associated security challenges. Although the SDN controller can provision global security policies throughout the network, it also tends to be a single point of failure. An attacker can either gain access to the controller or itself be a controller in order to control the entire network. An attacker can try to overcome the scaling limit of the controller to enforce its own control and deny services to the underlying forwarding devices and prospective applications. Although the programmability of SDN architecture is a major benefit to developers, attackers can easily tamper with the core functionality of the architecture [5]. Therefore, different compromised or malware infected applications can be installed on the controller, which can lead to unexpected network behavior. A virtual switch can be deployed at the edge of a network to scrutinize and filter out the incoming traffic in order to direct the suspicious flows toward mitigation and inspection devices. On the contrary, the same switch can be compromised by malware to inject erroneous flow or trigger distributed denial of service (DDoS) attacks [6]. Moreover, the communication path between SDN planes opens up a new frontier of security concerns. Any adversary can disrupt the communication path between the different planes in order to create a vulnerability void that can easily be compromised.

SDN Attack Vectors and Potential Solutions: Plane-Wise Analysis

Figure 2 shows the possible attacks at various layers of the SDN architecture. The plane-wise anal-

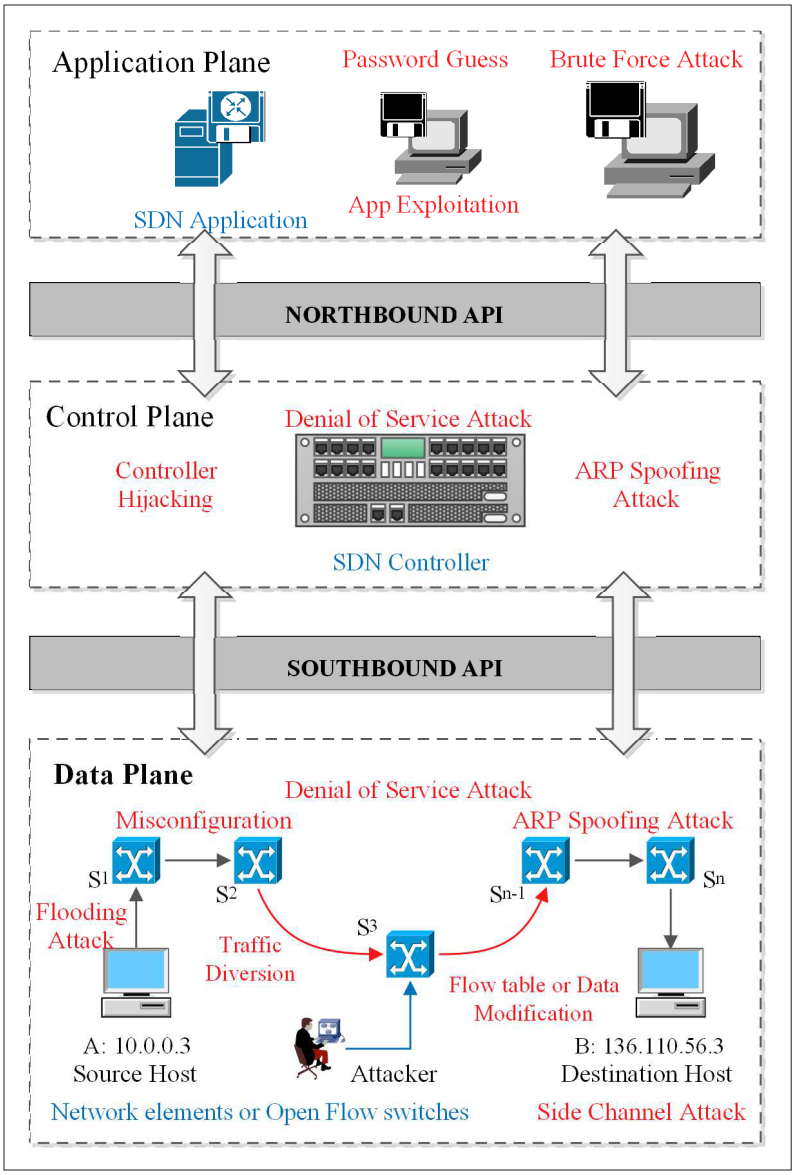

FIGURE 2. Attack vectors at different layers of SDN architecture.

ysis of these attack vectors and possible solutions are discussed below.

Application Plane and Northbound APIs

There are various applications and northbound application programming interfaces (API)s (Java, Python, REST, JSON, etc.) that an attacker can exploit in order to gain the network control through the controller. Accountability, access control, authorization, and authentication are major security issues that need to be addressed while designing security primitives for the application plane [7]. Some of the possible attacks at the application plane are described below [8–10].

Password Guessing or Brute Force: Using password guessing or brute force, an attacker can access the applications or northbound API, which may further lead to unauthorized access to the controller.

Application Exploitation: An unauthorized access to an application can be gained to perform illegal activities (malfunction, service disruption, or eavesdropping).

API Exploitation: A software component may contain malicious codes that can allow a hacker to perform illegal activities. API exploitation can be done at the northbound interface to disrupt the network flow.

Potential solutions for handling the above discussed attack vectors are presented below [8–10].

PermOF: A permission granting system for OpenFlow (OF) applications can be used to provide controlled access to the path and controllers using different permissions of read, notification, and write while allowing access to sensitive information.

Flover: A model checking system ensures that the OF policies do not collide with the network security policies.

Switch-Based Rule Verification (SRV): It is used to detect the malicious rules in the network topology.

Efficient Naming Scheme for Flow Entry: In this solution, the new entry is added in the OF table using two main elements: the security policy creating a person's role and the security privilege of that role.

Control Plane

The control plane of SDN is the prevalent choice of attackers for many reasons. For example, an adversary can target the controller to install new flow rules through different interfaces. If an adversary overtakes the controller, it can control the traffic at its will, thereby bypassing the security policies deployed at various planes. The attackers overpower the controller's scalability and control its availability to gain network control. Various attacks related to the control plane are described as below [8–10].

Controller Hijacking: Switches can be connected to a malicious controller as the IP address between the switch and the controller are not checked due to improper TCP connections between them.

DDoS Attack: An adversary floods a channel with unauthorized requests to halt its normal functioning.

ARP Spoofing Attack: An adversary can control the network illegally to steal or modify the data.

Network Manipulation: An attacker overtakes the SDN controller to produce false network data using which other chain attacks are initiated on the network.

The potential solution for handling these attack vectors at the control plane are presented below [8–10].

SDN-Guard: It is connected to the intrusion detection system at the control plane; it analyzes the incoming packet and informs about the probability of a threat.

Enhancing Scalability: Scalability can be improved by distributing the functions of the control plane, increasing the memory or processing power of the controller. For example, HyperFlow is a logically centralized but physically distributed control plane wherein decisions are made at the individual controller level [11].

Enhancing Availability: By modifying the OF model, the interaction between the control plane and the data plane is reduced. For example, in

DevoFlow, some control is given to the switches, while the main network control is assigned to the controller itself.

Data Plane and Southbound APIs

At this level, an adversary can control the network elements to install new flow rules in the flow tables or initiate DDoS attack [6, 11]. An attacker can easily compromise the forwarding devices to perform various attacks [8–10] discussed below:

- *Traffic diversion:* An attacker can control the network elements or forwarding devices to redirect the traffic flow and allow eavesdropping.
- *Side channel attack:* Latency detection between two elements in a channel can help the attacker know if there is any flow rule or not.
- *DoS:* An attacker floods a particular channel with unauthorized requests and packets that may disrupt the functioning of the complete transaction.
- *ARP Spoofing Attack:* Here, an attacker can control the network illegally to steal or modify the data.
- *Traffic sniffing:* It is used by a hacker to capture and analyze network communication information.
- *Data Modification:* During communication between the data plane and the control plane, data leakage is highly possible, and man-in-the-middle attack is used to gain complete access in the data plane.
- *Flow table modification:* In this attack, the flow tables installed at an OF switch are tampered with or modified.
- *Misconfiguration:* By making Transport Layer Security (TLS) optional in the OF switches, the chance of misconfiguration increases due to vulnerabilities.

The potential solution for handling the above discussed attack vectors at data plane are presented as below [8–10].

Veri-Flow: It is a tool placed between network elements and the SDN controller to detect malicious rules installed at the forwarding devices.

Network Planning: The topology of the network and distance between the switches and the controller must be taken into consideration. The minimum time a switch takes to transfer packets represents the lowest probability of being attacked as it becomes negligibly open for any attacker to view.

Mutual Authentication: Trust needs to be maintained between the control and data planes to avoid any attack. TLS and IPsec provide encrypted communication between the controller and switches, thereby providing integrity, confidentiality, and protection.

BlockSDN: Blockchain as a Service for SDN Attack Management

In this section, the blockchain process and its architecture is discussed followed by the attack scenarios generated at SDN layers. Moreover, the blockchain-based solutions for the mitigation of these attack scenarios are also described.

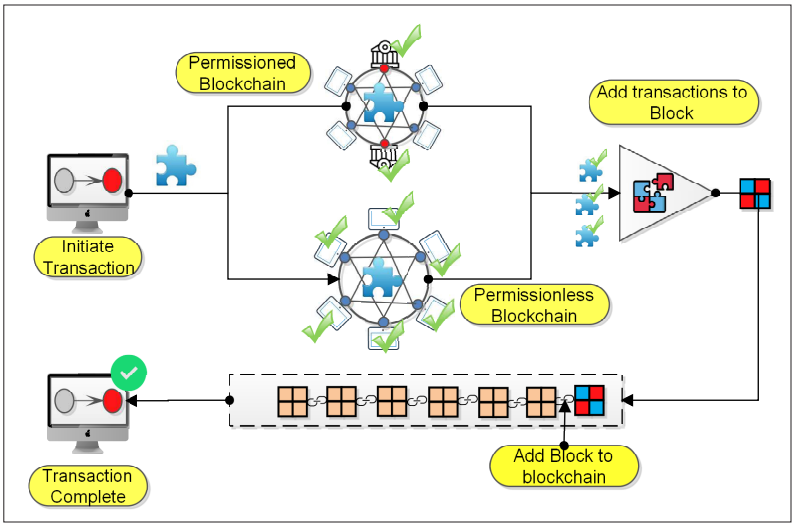

FIGURE 3. Blockchain: the process flow and architecture.

Blockchain: Basics, Architecture, and the Process

Whenever the thought of immutable distributed storage comes, the mind reflects back to the usage of *blockchain technology* [12]. Blockchain consists of blocks of data that are linked to each other using cryptographic hash functions. Each block of data constitutes some transactions that are recorded into it, a related Merkle root hash of the transactions (to maintain integrity), timestamps of the block creation, the hash of the block preceding the current block, and its own hash (for maintaining its integrity as a whole). Each block (before it is created) undergoes a process known as mining, which is undertaken by nodes in the blockchain called miner nodes. Every node has an equal opportunity of becoming a miner. The miners, upon successful mining of a block, receive some incentives as a reward for participating in the mining process, which is based on the consensus mechanism [13].

Numerous consensus methods have been proposed from time to time. For example, in proof of work, a miner has to prove its legitimacy by solving some complex mathematical problem that consumes resources. In proof of stake, the miner has to prove its stake holding to do the mining. Similarly, there are many other methods designed for mining purpose that have some pros and cons associated with them. Thus, depending on the kind of problem being dealt with, an appropriate mining mechanism can be selected [14]. Once a block is mined, it is added to the existing blockchain available with all the participating nodes. Now, if at any node a block is manipulated, the blockchain is in an error state on that specific node, and therefore data tampering can be identified. The process flow and architecture of blockchain transactions is depicted in Fig. 3. With the evolution of different applications in smart cities, two different variants of blockchain, public/permissionless and private/permissioned blockchains, have evolved over time.

Permissionless or Public Blockchain: This is the most common type of blockchain, used for famous cryptocurrency like bitcoin. Anyone can be part of this blockchain, and anyone can become a miner. It is a simple and very powerful

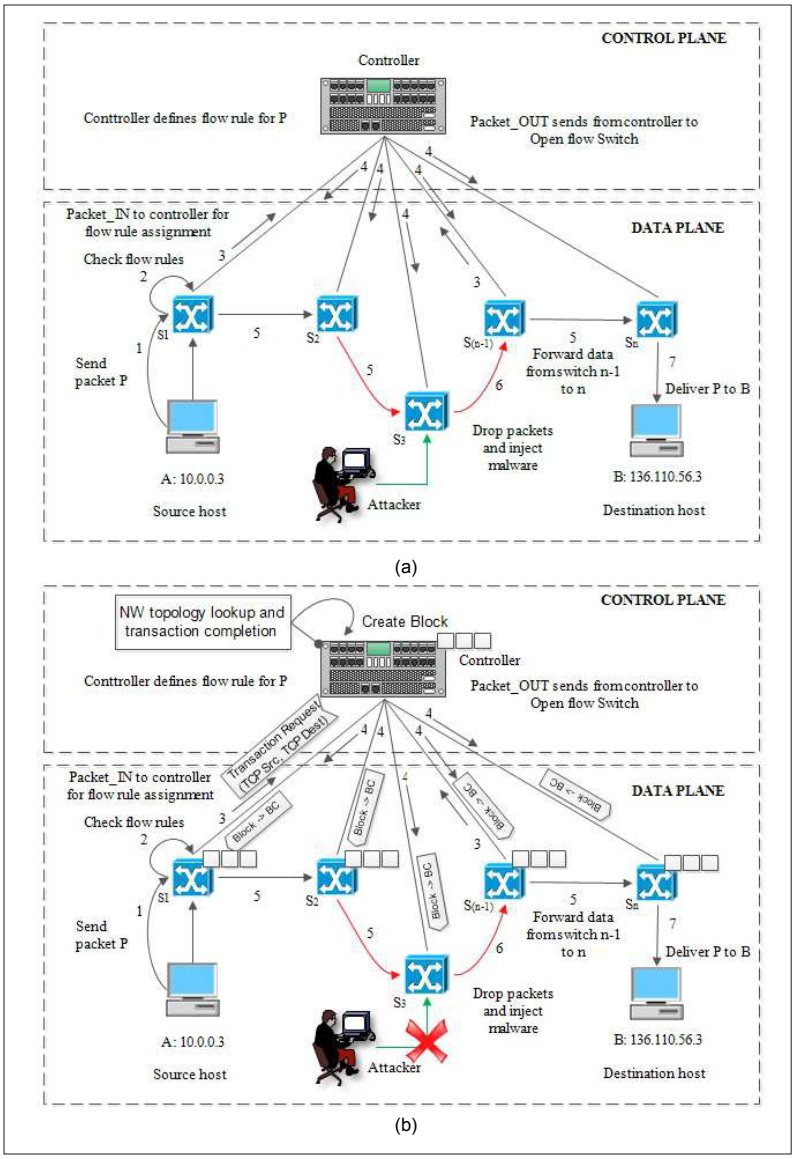

FIGURE 4. Malware-compromised switch in SDN architecture; a) Case1: malware-compromised switch; b) Case2: blockchain as a service for secure traffic flow.

blockchain as it can scale up. But the scalability becomes a problem for this kind of blockchain as anyone can be part of the blockchain, so there is no control on whom to allow access and whom to restrict.

Permissioned or Private Blockchain: Contrary to the actual blockchain idea (i.e., to create a truly decentralized storage solution), permissioned or private blockchains have evolved as a solution to address the issue of access control and authentication/authorization. The semidistributed blockchain variant popularly known as the permissioned blockchain has a central authority that controls the entry and access of the nodes in the blockchain. It helps in filtering out potential practitioners.

Blockchain as a Service for Mitigation of Malware Compromised Switch Attack at the Data Plane

The most common attack at the data plane is malware injection via a compromised switch that is part of the normal flow. This attack scenario is depicted in Fig. 4a, which considers the case where host A (10.0.0.3) tries to send data packets to host B (136.110.56.3). In an ideal case of data flow, initially host A sends a packet (P) to the OFswitch (S1) to which it is connected directly. On receiving P from host A, the initial inspection of the packet is done at S1. According to the packet characteristics, the flow table is scrutinized to find a suitable matching entry. If a matching flow entry is available in the flow table, the corresponding action is performed. However, as depicted in the attack scenario, the matching flow entry was not found, so S1 sends a Packet IN request to the controller for installing a new flow rule. After this, the controller inspects P and checks the entire network topology to establish a suitable flow rule. Accordingly, it sends the flow rule in the form of Packet OUT to all the data plane devices that would be part of the transmission process. Eventually, the packet that started from host A will now pass through devices S1, S2, ..., S*n* to reach host B. This ideal scenario ends up with the consideration that any of these devices may be compromised. An attacker may break into any of the OF switches and then inject its own traffic, thereby dropping the original packet. In Fig. 4b, S3 is shown as a malware-compromised switch being controlled by an attacker. Now, ideally S3 is supposed to relay the incoming packets to S4; however, it is sharing the data to the attacking node. In this case, the attacker node is also injecting its own traffic into the flow while dropping the original packets.

To deal with this problem, the flow tables and the packets of the flow are secured using permissioned blockchain. Tasks like creating a new block and adding to a blockchain are limited to the controller in a permissioned blockchain manner. The consensus mechanism is realized using a proof of stake mechanism wherein the controller stakes its network topology repository [14]. As shown in Fig. 4b, the process is initiated when host A starts transmission to host B. Once the message reaches the directly connected S1, it checks the blockchain for the associated flow rule. If the rule is not found in any block, the request is forwarded to the controller with the sender's and receiver's IP addresses. The controller (the miner for permissioned blockchain) collects all transactions coming to it since the creation of the previous block. Then it validates them based on the network topological view and adds the transaction into a new block. This new block is sent to all OFswitches under the control of the controller. The OFswitches uses the information from the blockchain and performs the action (i.e., forward, enqueue, drop, or modify). Now, if an attacker tries to capture packets and insert its own packets into the ongoing flow, the Merkle root hash gets changed as the source address of the newly added packets will be that of the attacker now. The blockchain immediately indicates that the hashes of some blocks are not matching the chained blocks, thereby preventing the attack.

The proposed blockchain architecture for handling the problem of a malware-compromised switch using permissioned blockchain is shown in Fig. 5. The sequence of activities for the proposed scheme are described as:

- A switch (S) sends a request for joining the

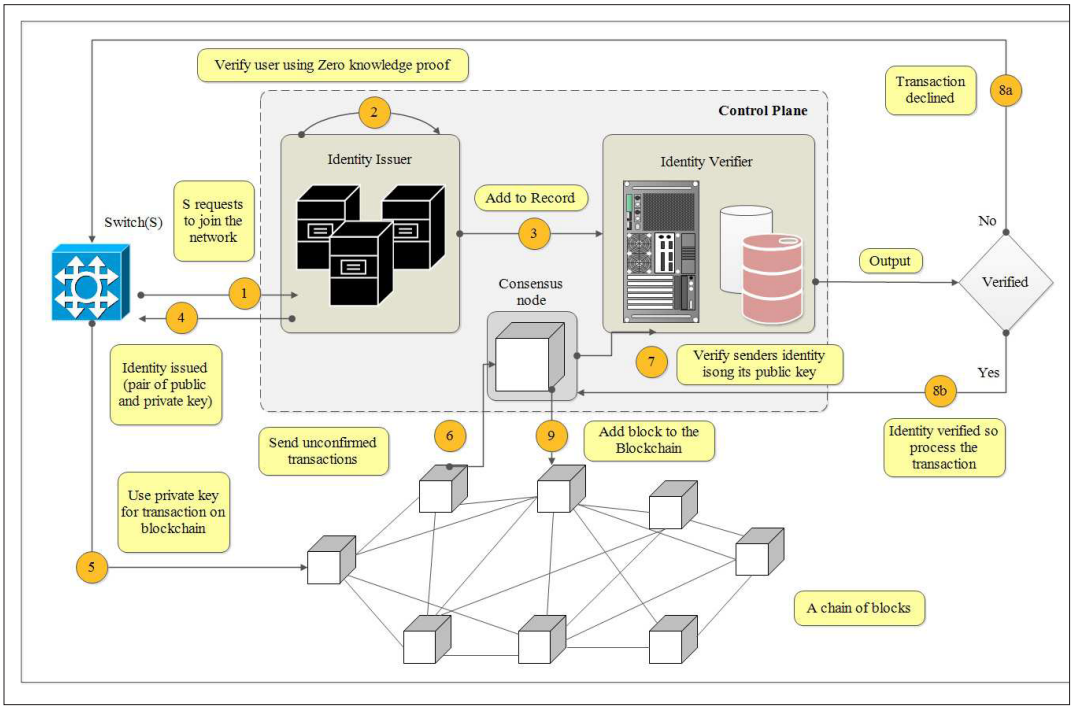

FIGURE 5. Blockchain as a service for switch verification and validation.

blockchain network to the controller, which passes it to the identity issuer located at the control plane.

- The purpose of the identity issuer is to verify the authenticity of a switch and then issue an identity (SID) that is valid for the considered blockchain.
- The identity issuer verifies the switch based on the authentication information provided using the zero knowledge proof concept. The zero knowledge proof is the method in which one party can prove its knowledge about some information X to another party, without revealing the value of X. The two communicating parties are referred to as prover and verifier. If the prover provides additional information about actual facts to the verifier, the zero knowledge proof structure is broken [15]. If the considered switch is successfully verified using zero knowledge proof, it is added to the list of verified switches in the identity verifier's database.
- On successful addition to the database, a pair of public and private keys are issued to the switch. The public key is also provided to all the switches that are part of the blockchain.
- On receiving the keys, the requesting switch can carry out the transactions (eg., flow table update) on the blockchain by encrypting them using its own private key.
- Each member of the blockchain can now verify the switch that has issued the transaction. However, since the transaction is still not added to the blockchain, it is of no use yet. It has to be added into the blockchain by the miner or consensus node that resides at the control plane. The consensus node fetches all the unconfirmed transactions, which are still not added into the blockchain.
- Upon receipt of an unconfirmed transaction, the consensus node sends it to the identity verifier for verification.
- If the key is verified by the identity verifier, the information is sent to the consensus node for further processing of the transaction. Otherwise, the transaction is declined, and the same is informed to the requesting switch.
- Now, the consensus node adds the transaction to the block being created at time *t*. Once the block is created, it is synchronized to all the switches that are part of the permissioned blockchain. Now, each switch has the flow rules residing in the blockchain. If an attacker tries to inject some flow rule that is not legitimate, the modification will simply be reflected in the blockchain as the hash value of the blocks created thereafter will not match previous block hashes, thereby breaking the fundamental linking of the blockchain.

Blockchain as a Service for Mitigation of DDoS Attack at the Control Plane

Among various SDN attacks, DDoS/DoS is one of the more popular kind. The attack scenario for generating a DDoS attack at the control plane of SDN through a malware-injected switch is shown in Fig. 6a. The network topology depicted for the attack scenario consists of three levels of switches: 0, 1, and 2. The attack scenario proceeds in the following manner:

- An attacker captures the switch at level 0 through a malware injection.
- The attacker starts a packet transfer request and sends it to switch AS1.
- Since there is now a matching flow available, AS1 sends a Packet IN request to the controller for the flow information regarding the packet in question.
- The controller sends the flow details to the

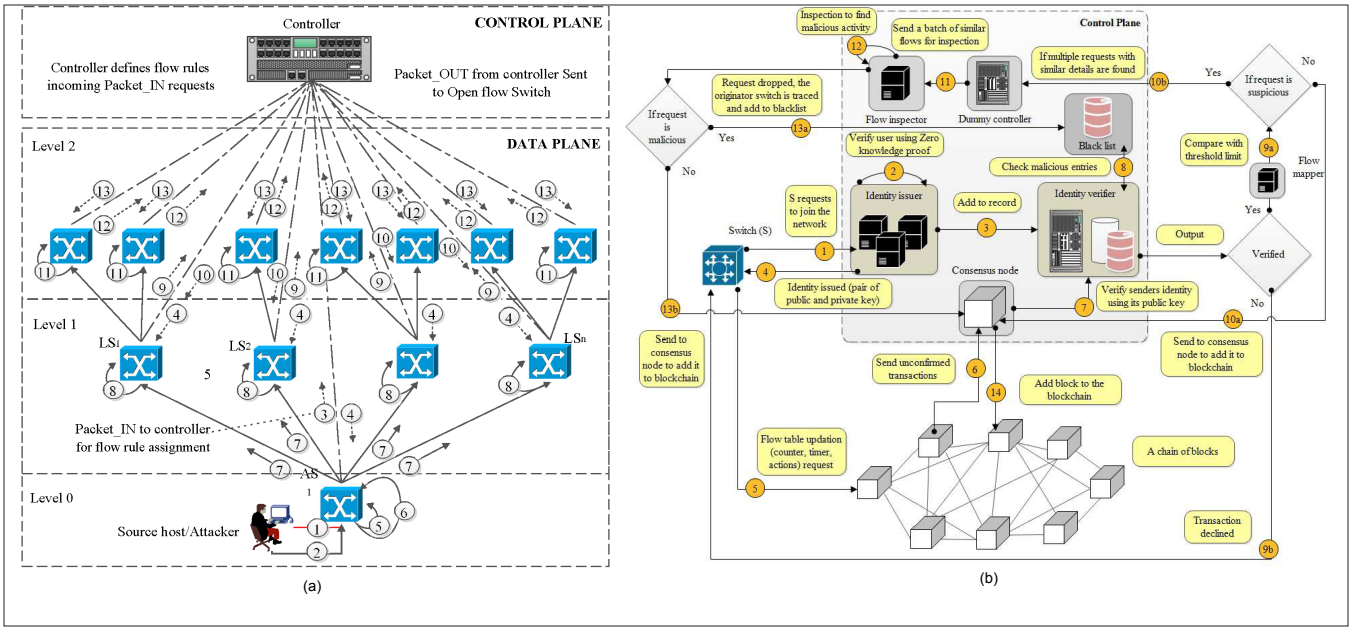

FIGURE 6. Generation and mitigation of DoS/DDoS attack: a) DoS/DDoS attack generation; b) blockchain as a service for DoS/DDoS prevention.

switches that are expected to be part of the flow (i.e., AS1 and switches LS1, LS2, ..., LS*n* located at level 2).

- The malicious script inserted into switch AS1's operating system alters the flow information received from the controller, changing the action type to forward; flood.
- The timer of the flow entry received from the controller at AS1 and level 1 switches is allowed to expire; then the attacker floods the received packet to the immediate neighbors.
- The packet comprising a trojan is forwarded to the switches at level 1.
- When the packet comprising the trojan reaches the switches at level 1, the flow rule has already timed out and no longer exists on level 1 switches.
- Now, each switch located at level 1 sends Packet IN request to the controller.
- The controller sends the Packet OUT containing the new flow entry to all switches at level 1 and the switches expected to be part of the flow.
- Now, the trojan alters the action field of the flow table at the level 1 switches and floods it to the neighboring level 2 switches in a similar way as was done in the fifth step.
- The level 2 switches, upon receiving the packets, send Packet IN requests to the controller.
- The controller again calculates the path and sends Packet OUT to all level 2 switches and the switches that are expected to be part of the flow.

In this way, the malicious packet keeps on propagating through the data plane level by level, and the magnitude of the attack grows exponentially, leading to the choking of the controller.

The proposed blockchain-as-a-service architecture to handle the DDoS attack is illustrated in Fig. 6b. The sequence of activities are described below.

- Initially, S requests to join the permissioned blockchain network, and the request is passed to the identity issuer.
- The identity issuer verifies the authenticity or legitimacy of the switch using zero knowledge proof and then issues an identity (SID).
- If the verification is successful, S is added to the list of verified switches in the verifier's database.
- After addition to the database, a pair of public and private keys are issued to S. The public key is also made available to all the switches that are part of the blockchain.
- Upon receiving the keys, the requesting switch initiates the transactions for flow table update (e.g., update of counter, timer, or action field for DDoS generation) on the blockchain by encrypting them using its own private key.
- Each member of the blockchain verifies the switch that has issued the transaction. However, since the transaction is still pending, it has to be added into the blockchain by the miner or consensus node. For this reason, the consensus node fetches all the unconfirmed transactions that have yet to be added into the blockchain.
- The consensus node sends the unconfirmed transactions to the identity verifier for authentication.
- The identity verifier checks the blacklist log, which holds the malicious flows and switches' identities.
- If the unconfirmed transactions and the associated switch identity are not available in the blacklist log, the transaction is approved, and the control is passed to the flow mapper for replication analysis. On the contrary, if the unconfirmed transactions and the associated switch identity are available in the blacklist, the transaction is declined.
- The flow mapper is used for replication analysis and compares the incoming flows

from various switches with the threshold limit set by the controller. If a similar Packet IN request is coming from more than one switch at the same time, such requests are forwarded to a dummy virtual controller; otherwise, they are forwarded to the consensus node for further processing.

- A dummy controller is deployed to fool the attacker as it makes the attacker presume that the transaction has reached the actual controller. It also relieves the central controller from the load related to the suspicious requests. It checks all the incoming flow transactions and adds the multiple requests that are related to similar flow attributes to a batch. Now, this batch of similar flow requests is sent to a flow inspector for further inspection.
- The flow inspector inspects the attributes of the requests for potential threat detection. If the batch of requests is malicious, they are dropped.
- Once the incoming requests are identified as malicious, the question arises as to whether these requests are generated by a single device or multiple devices. If all such requests are generated to the controller from a single device (DoS attack), this originator node is traced. The flow inspector checks for the actual originator node for these flows. Once identified, these originator switches are added to the blacklist log. In another case, there may be different forwarding devices that have generated the requests to the controller (DDoS). In such a case, there has to be some machine that is controlling the entire process. Such a machine or device is identified by the controller and then added to the blacklist log. In this way, all the switches working under the control of an attacking device are relieved, and they can perform normal activities. If the requests are not malicious, such information is sent to the consensus node for further processing of the transaction.
- Now, the consensus node adds the transaction to the block being created at time *t*. Once the block is created, it is synchronized to all the switches that are part of the permissioned blockchain. Each switch will have the flow rules residing in the blockchain available to it.

Open Issues and Challenges

Although the BlockSDN framework is capable of mitigating various attacks at SDN planes, there are still many open issues and challenges that must be handled effectively in the near future to make this framework more robust and reliable. The identified open issues and challenges are discussed below.

Storage: Storing blockchain in OF switches requires memory resources in large numbers, which can become a bottleneck as the size of blockchain increases.

Computational Load: The controller will have to manage the operations related to mining, validity issuing, and verification, which require superior computations.

Delay and Response Time: The proposed scheme enables addition of flow tables to blockchain, which introduces delay in comparison to the line speed.

The proposed solution is not limited to a single controller and uses two controllers: an actual controller and a dummy controller (to foil the attacker). However, a deep analysis of the proposed solution is still required in a multi-controller scenario to understand the associated implications.

Centralized Miner: Permissioned blockchain can bring the network to one point at which it could fail. In future work, the mining work can be distributed over multiple controllers to enable decentralization.

Computational Complexity: The load of mining, validity issuing, and verification at the controller would eventually lead to an increase in the computational complexity.

Computational Cost: The additional use of computational resources can lead to an increase in the computational cost, which is not desired by any effective solution.

Interoperability: As the hardware devices and switches are not vendor-specific, different issues may arise related to interoperability that need to be handled effectively using appropriate mechanisms. Even more, interoperability issues related to different blockchains should be handled carefully.

Multi-Controller Scenario: The proposed solution is not limited to a single controller and uses two controllers: an actual controller and a dummy controller (to foil the attacker). However, a deep analysis of the proposed solution is still required in a multi-controller scenario to understand the associated implications.

Conclusions

Smart city infrastructure is facing a tough challenge to provide reliable connectivity to handle the broad range of applications. The dependency of conventional network architecture on standard protocols and algorithms puts a limitation on the scalability and flexibility of service-oriented architecture. To overcome these issues, SDN is widely adopted in industrial and commercial environments for provisioning of reliable and dynamic flow routes. However, besides being the top choice of network administrators, SDN has to deal with several security issues, which have opened the door to new research verticals. The dependency on a logically centralized controller adds to the concern as it is a popular target for attackers. In this article, a concise analysis on various prominent attack vectors for the SDN architecture is presented. From this analysis, it can be concluded that the SDN architecture has to prevent vulnerabilities not only at different planes but also at the intermediate communication paths. Moreover, different possible solutions to cope with the discussed attack vectors are discussed. The consistency and integrity of flow entries have to be maintained in order to prevent manipulation of traffic routes by attackers using malware codes. Two different cases of malware injection at the data and control planes have been described. In the first case, a malware-infected switch can inject its own traffic while dropping the original packets. In the second case, a malware-infect-

Besides being the top choice of network administrators, SDN has to deal with several security issues that have opened the door to new research verticals. The dependency on a logically centralized controller adds to the concern as it is a popular target for attackers.

ed switch can insert malicious script on various neighboring switches in order to trigger DDoS attack at the control plane. To prevent such attack scenarios, blockchain-based mitigation techniques have been designed in this article. Due to the centralized architecture of SDN, the permissioned blockchain model is used for the design of these techniques. Blockchain is an emerging technology that can be exploited further to handle different security concerns of the SDN architecture. Finally, the open issues and challenges that have come up due to the amalgamation of blockchain with SDN have been highlighted.

Acknowledgments

This work is partially supported by the National Key Research and Development Program, No. 2017YFE0125300, the National Natural Science Foundation of China-Guangdong Joint Fund under Grant No. U1801264, the Jiangsu Key Research and Development Program, No. BE2019648, and the Melbourne-Chindia Cloud Computing (MC3) Research Network.

References

[1] J. Gubbi *et al.*, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013, pp. 1645–60.
[2] C. V. Networking, "Cisco Global Cloud Index: Forecast and Methodology 2015–2020," white paper, 2016; https://www.cisco.com/c/dam/m/en-us/serviceprovider/ cisco-knowledgenetwork/files/622-11-15-16-Cisco-GCICKN-2015-2020-AMER-EMEAR-NOV2016.pdf, accessed Jan. 2019.
[3] G. S. Aujla *et al.*, "Data Offloading in 5G-Enabled Software-Defined Vehicular Networks: A Stackelberg-Game-Based Approach," *IEEE Commun. Mag.*, vol. 55, no. 8, Aug. 2017, pp. 100–08.
[4] A. Jindal *et al.*, "Sedative: SDN-Enabled Deep Learning Architecture for Network Traffic Control in Vehicular Cyber-Physical Systems," *IEEE Network*, vol. 32, no. 6, Nov./Dec. 2018, pp. 66–73.
[5] Q. Li *et al.*, "Security Policy Violations in Sdn Data Plane," *IEEE/ACM Trans. Networking*, vol. 26, no. 4, 2018, pp. 1715–27.
[6] S. Deng *et al.*, "Packet Injection Attack and Its Defense in Software-Defined Networks," *IEEE Trans. Info. Forensics and Security*, vol. 13, no. 3, 2017, pp. 695–705.
[7] C. Yoon *et al.*, "Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks," *IEEE/ACM Trans. Networking*, vol. 25, no. 6, 2017, pp. 3514–30.
[8] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Commun. Surveys & Tutorials*, vol. 18, no. 1, 2015, pp. 623–54.
[9] T. Dargahi *et al.*, "A Survey on the Security of Stateful SDN Data Planes," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 3, 2017, pp. 1701–25.
[10] A. Akhunzada and M. K. Khan, "Toward Secure Software Defined Vehicular Networks: Taxonomy, Requirements, and Open Issues," *IEEE Commun. Mag.*, vol. 55, no. 7, July 2017, pp. 110–18.
[11] S. Gao *et al.*, "Security Threats in the Data Plane of Software-Defined Networks," *IEEE Network*, vol. 32, no. 4, July/Aug. 2018, pp. 108–13.
[12] S. Aggarwal *et al.*, "Energychain: Enabling Energy Trading for Smart Homes Using Blockchains in Smart Grid Ecosystem," *Proc. 1st ACM MobiHoc Wksp. Networking and Cybersecurity for Smart Cities*, 2018, p. 1.
[13] S. Tuli *et al.*, "Fogbus: A Blockchainbased Lightweight Framework for Edge and Fog Computing," *J. Systems and Software*, vol. 154, 2019, pp. 22–36.
[14] A. Jindal, G. S. Aujla, and N. Kumar, "Survivor: A Blockchain Based Edge-as-a-Service Framework for Secure Energy Trading in Sdn-Enabled Vehicle-to-Grid Environment," *Computer Networks*, vol. 153, 2019, pp. 36–48.
[15] P. Matias *et al.*, "NIZKCTF: A Noninteractive Zero-Knowledge Capture-The-Flag Platform," *IEEE Security Privacy*, vol. 16, no. 6, Nov 2018, pp. 42–51.

Biographies

Gagangeet Singh Aujla [S'15, M'18] (gagi_aujla82@yahoo.com) is working as a postdoctoral research associate in the School of Computing at Newcastle University, United Kingdom. Previously (2018–2019), he was an associate professor in the Computer Science and Engineering Department, Chandigarh University, India. Prior to this, he was a research associate (2017–2018) in an Indo-Austria research project sponsored by the Department of Science and Technology, Government of India and the Ministry of Science, Austria. He received the 2018 IEEE TCSC Outstanding Ph.D. Dissertation Award, which recognized his leading expertise in the application of scalable and sustainable algorithms for cloud data centers, software defined networks, and smart grid.

Maninderpal Singh Aujla [S'19] (mpvirdi@gmail.com) is working toward his Ph.D. in the Computer Science and Engineering Department, Chandigarh University, India, where is also an assistant professor. He received his M.Tech. from the Computer Science and Engineering Department, Lovely Professional University, India, in 2013. He received his B.Tech. from the Computer Science and Engineering Department, Punjab Technical University, India, in 2010.

Arnab Bose (arnabmy@live.com) received his M.Tech in computer science and engineering from Chandigarh University. He worked as a senior research fellow at the Defence Research and Development Organisation, Dehradun,India, from November 2013 to August 2017. He received his B.Tech. in computer science and engineering from Himachal Pradesh University, India, in 2010.

Neeraj Kumar [M'16, SM'17] (neeraj.kumar@thapar.edu) is a professor in the Department of Computer Science and Engineering, Thapar Univesity. He received his M.Tech. from Kurukshetra University, India, followed by his Ph.D. from SMVD University, Katra, in computer science and engineering. He was a postdoctoral research fellow at Coventry University, United Kingdom. He has more than 250 research papers in leading journals and conferences of repute. He is a Technical Editor of *IEEE Network* and an Associate Technical Editor of *IEEE Communication Magazine*. He is an Associate Editor of *IJCS*, Wiley, *JNCA*, Elsevier, and *Security and Communication*, Wiley, and on the Editorial Board of *Computer Communications*, Elsevier.

Guangjie Han [S'03–M'05–SM'18] (hanguangjie@gmail.com) received his Ph.D. degree from Northeastern University, Shenyang, China, in 2004. He worked as a postdoctoral eesearcher with the Department of Computer Science, Chonnam National University, Gwangju, Korea. He was a visiting research scholar with Osaka University, Suita, Japan. He was a visiting professor at City University of Hong Kong, China. He is currently a professor with the Department of Information and Communication System, Hohai University, Changzhou, China. He is also a professor with the College of Engineering, Nanjing Agricultural University, Nanjing, China.

Rajkumar Buyya (rbuyya@unimelb.edu.au) is currently a Redmond Barry Distinguished Professor and the Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He also serves as the founding CEO of Manjrasoft Pty. Ltd. His research interests include cloud computing, fog computing, and parallel and distributed systems.