Quantifying and Querying Network Reachability

Amir R. Khakpour Alex X. Liu
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824
{khakpour, alexliu}@cse.msu.edu

***Abstract*—Quantifying and querying network reachability is important for security monitoring and auditing as well as many aspects of network management such as troubleshooting, maintenance, and design. Although attempts to model network reachability have been made, feasible solutions to computing network reachability have remained unknown. In this paper, we propose a suite of algorithms for quantifying reachability based on network configurations (mainly ACLs) as well as solutions for querying network reachability. We present a comprehensive network reachability model that considers connectionless and connection-oriented transport protocols, stateless and stateful routers/firewalls, static and dynamic NAT, PAT, etc. We implemented the algorithms in our network reachability analysis tool called Quarnet and conducted experiments on a university network. Experimental results show that the offline computation of reachability matrices takes a few hours and the online processing of a reachability query takes 0.075 seconds on average.**

I. Introduction

A. Motivation

Although computer networks are created to provide reachability between end hosts, various sophisticated mechanisms, such as router Access Control Lists (ACLs) and firewalls, have been deployed to limit such reachability for security or privacy purposes, and various factors, such as routing dynamics and network address translation, may affect such reachability, in unexpected ways. As a critical infrastructure of national importance, the Internet faces unprecedent challenges for reliable reachability. Correctly implementing the exact reachability that is needed for a large interconnected network is crucial. While more reachability than necessary may open doors to unwanted and even malicious traffic causing sometimes irreparable damages, less reachability than necessary may disrupt normal businesses causing huge revenue losses.

Unfortunately, in reality, network reachability management is often a mess for most networks due to its high complexity and the lack of advanced reachability debugging tools. First, network configurations have become far more complex than even a skilled operator can correctly manage. The complexity of modern networks has been rapidly increasing due to the explosive growth of Internet connectivity expanding from end-hosts to pervasive devices and network supported applications of various scales. Second, due to the lack of advanced reachability debugging tools, the current common practice for reachability management is still "trial and error", which of course results in a plethora of reachability errors. Configuration errors have been observed to be the largest cause of failure for Internet services [14]. A recent Yankee Group report has shown that more than 62% of network downtime is due to human configuration errors and more than 80% of IT budgets are allocated towards maintaining just the status quo [7]. Network operators face tremendous pressure to fix problems quickly because operational networks often support critical business applications and important communications and the loss caused by network outrages becomes increasingly acute. For example, the estimated revenue losses per hour of downtime for the industry of media, banking, and brokerage are 1.2, 2.6, and 4.5 million dollars, respectively [7]. Quantitative studies have shown that most firewalls are misconfigured [18]. The reality could be worse than these published staggering numbers as the errors of more reachability than needed are often go undetected, while less reachability than needed is a common source of complaints to operators. Therefore, a scientific approach, instead of "trial and error", is needed to manage network reachability, which will continue to grow more complex as networking technologies evolve.

As organizations continually transform their network infrastructure to maintain their competitive edge by deploying new servers, installing new software and services, expanding connectivity, etc, operators constantly need to manually modify their network configurations. Ideally, such manual modifications should exactly implement the desired changes; however, practically, they often come with undesirable, yet unnoticed, side effects on reachability. Therefore, an automated tool that monitors the all-to-all reachability of a large interconnected network in realtime is crucial to its successful management. Furthermore, querying and verifying reachability is a routine, yet error-prone, task for operators. An automated reachability query and verification tool is needed.

To enforce the right amount of network reachability, no more and no less, for a large interconnected enterprise network to achieve access control, security, and privacy, in this paper, we investigate two aspects of network reachability: quantification and querying, and present a reachability management tool, which we call Quarnet. Quantifying the reachability between any two subnets means to compute the set of packets that can traverse from one subnet to another based on network configurations. Querying reachability means to ask questions like "what can access what". They are useful in many aspects of network management:

(1) *Network Security Monitoring and Auditing*: Verifying that the deployed ACLs satisfy certain security specifications is an integral part of network security monitoring and auditing. The current practice is to send probing packets. However, this approach has drawbacks. First, it is infeasible to generate all possible probing packets. Second, as routing tables change over time, the auditing result is valid only for a specific time. Combining the current active probing approach with quantitative network reachability analysis, we can have comprehensive security auditing for large enterprise networks.

(2) *Network Troubleshooting*: An important task for network administrators is to troubleshoot reachability problems when two hosts fail to communicate or there is unauthorized traffic passing through a series of ACLs. For large complex networks, troubleshooting reachability problems is extremely difficult. To

check the reachability of a path, the current practice is to check the reachability of every hop in the path by actively sending probing packets. This approach is disruptive, time consuming, and sometimes infeasible when isolating some hops is impossible. In contrast, when the reachability of every path has been pre-computed, troubleshooting reachability problems and identifying faulty ACLs is easy.

(3) *Network Security Design, Maintenance, and Reachability Monitoring*: Designing ACLs based on certain security policies is an important part of network design. Before deploying a network, it is important to first verify reachability. This helps to avoid security breaches and service outbreaks caused by misconfigured ACLs. Furthermore, networks change over time with the evolving of topology and connected servers/hosts. Often network changes require corresponding changes on ACL rules. Due to the interconnected nature of networks, a modification on one ACL may have unnoticed side effects on network reachability such as causing two servers to fail to communicate or opening security holes by enabling unauthorized accesses. Thus, network reachability analysis helps to detect and resolve potential problems before committing any change to ACLs.

B. Technical Challenges

Computing network reachability is hard. First, from the reachability perspective, the interaction among the rules in one ACL is already complex due to the multi-dimensionality of ACL rules, but the interaction of multiple ACLs in interconnected networks is even more complex. Second, routing and network topology have complex impact on reachability. There is typically more than one path between a source and a destination, and a packet may only be able to traverse from the source to the destination via some, but not all, available paths. Third, middleboxes often have complex impact on reachability. For example, packet transforming (such as NAT and PAT) middleboxes complicate reachability calculation because they modify packets headers when they are traveling in a network. Fourth, transport layer protocols also complicate reachability calculation because for connection-oriented protocols the reachability of both data path and signalling path should be taken into account. It is even more challenging while there are some stateful middleboxes in the path. Last but not least, the problem space is huge as the ACL rules are typically specified over the standard 5-tuple, which have 104 bits in total.

C. Our Approach

In this paper, we present Quarnet, a tool that comprises a suite of concrete algorithms for quantifying and querying network reachability. For quantification, Quarnet takes a network topology and the ACLs deployed on middleboxes as its input, and outputs reachability matrices that represent the lower-bound reachability (*i.e.*, the minimal set of packets that can traverse from a source to a destination at any time), instantaneous reachability (*i.e.*, the set of packets that can traverse from a source to a destination at a particular time), and upper-bound reachability (*i.e.*, the maximal set of packets that can traverse from a source to a destination at some time) for every pair of source and destination subnets. For querying, Quarnet allows network operators to ask questions about the reachability of the subnets in their network, *e.g.*, "which hosts in a subnet can access the mail server in another subnet?". We proposed a language for formally specifying reachability queries. To efficiently process queries, we use decision diagrams as the data structure for representing reachability matrices.

Quarnet can be deployed on a server as shown in Figure 1. Initially, this server first collects the configuration file from each middlebox (via SNMP for example) and the network topology from the administrator and then computes reachability matrices offline. Afterwards, network operators can perform reachability queries through a GUI interface, where each query is then formulated by an SQL-like query language and processed by the Quarnet query module. Fine-grained access control to reachability matrices can be enforced so that different operators can perform different reachability queries. Each time the configuration of a middlebox or the topology of the network is changed, the Quarnet server needs to be notified and reachability matrices need to be updated accordingly.

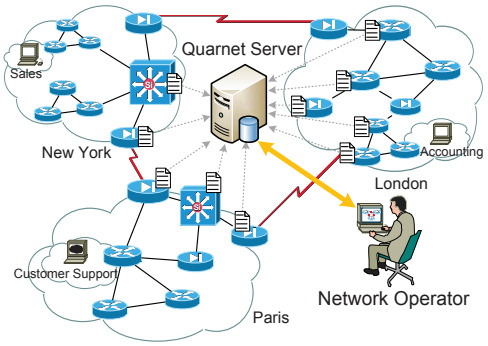

Fig. 1. Quarnet architecture

D. Key Contributions

We make four key contributions in this paper. (1) We propose a comprehensive reachability modeling and formulation that encompasses elements that have not been addressed in prior work such as dynamic NAT, PAT, connection orientation of transport protocols, and statefulness of middleboxes. (2) We propose efficient algorithms for computing network reachability. (3) We propose a language and solutions for querying network reachability. (4) We implemented Quarnet in C++ and experimented on a university network. Experimental results show that the offline computation of reachability matrices takes 11 hours and the online processing of a reachability query takes an average of 0.075 seconds.

II. Reachability Modeling and Formulation

In this section, we first introduce a network model, based on which we formulate three network reachability metrics: *lower-bound reachability*, *instantaneous reachability*, and *upper-bound reachability*. We also differentiate network reachability formulations based on transport protocol types and the presence of network address translation.

A. Network Modeling

In this paper, we model a network as a non-simple directed bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of arcs over $\mathcal{V}$, and $\mathcal{F}$ is a set of ACLs. Each vertex in $\mathcal{V}$ represents a subnet or a middlebox. We use the term "subnet" to represent a set of adjacent subnetworks (*i.e.*, local area networks (LANs) or VLANs, where either they have the same reachability (*i.e.*, there is no ACL deployed between any two subnetworks in the set) or the reachability among the subnetworks is not a concern. For example, given an enterprise network, we represent the outside Internet as a subnet. The term "middlebox" refers to any networking device that can

forward packets from one subnet to another, such as a network router, a firewall, a traffic shaper, or a L3 switch. Let $\mathcal{N}$ be the set of subnets in $\mathcal{V}$ and $\mathcal{R}$ be the set of middleboxes in $\mathcal{V}$. Each arc in $\mathcal{E}$ represents a unidirectional physical link between a subset in $\mathcal{N}$ and a middlebox in $\mathcal{R}$. Each ACL in $\mathcal{F}$ filters the packets traveling on an arc in $\mathcal{E}$.

We model network links as unidirectional arcs because every ACL is associated with a unidirectional physical link and each bidirectional link can be modeled as two unidirectional arcs. Note that some physical links, such as satellite links, are physically unidirectional. We model a network as a bipartite graph because between two adjacent subnets there is at least one middlebox and between any two middleboxes there exists at least one subnet. We model a network as a non-simple graph because between a subnet and a middlebox there may exist multiple physical links for backup.

Given a network with m middleboxes, where the maximum number of unidirectional interfaces on a middlebox is denoted h, we represent the network by an $m \times h$ matrix $\mathcal{I}$ called the *Network Incident Matrix*. Here $\mathcal{I}[i,j] = N$ if and only if subnet N connects to middlebox i on its interface j, and $\mathcal{I}[i,j] = 0$ if and only if no subnet connects to middlebox i on its interface j. For simplicity, incoming interfaces are represented by even numbers and outgoing interfaces are represented by odd numbers. Similarly, we represent the ACLs deployed on the network by an $m \times h$ matrix $\mathcal{A}$ called the *ACL Matrix*. We use $\mathcal{A}[i,j]$ to denote the ACL deployed on the j-th interface of middlebox i.

Figure 2(a) shows a network with three middleboxes and four subnetworks. Two VLANs (S1 and S2) are connected to an L3 switch (SW). One subnetwork (S3) and a DMZ (S4) are connected to firewall (FW). SW and FW are connected to the Internet through a gateway router (GW). Figure 2(b) shows graph representing the topology. Note that we assume there is an ACL on each interface of the middleboxes. The graph consists of 11 vertices representing the 8 subnets S1, ..., S8 and the 3 middleboxes SW, FW, and GW. Note that S1, ..., S4 denote the four subnetworks LAN1, LAN2, LAN3, and DMZ, S5 denotes the outside Internet, and S6, ..., S8 denote the subnetworks that connects two adjacent middleboxes. The network incident matrix for this network with $m = 3$ and $h = 8$ is :

$$\mathcal{I} = \begin{pmatrix} S1 & S1 & S2 & S2 & S6 & S6 & S7 & S7 \\ S3 & S3 & S4 & S4 & S7 & S7 & S8 & S8 \\ S5 & S5 & S6 & S6 & S8 & S8 & 0 & 0 \end{pmatrix}$$

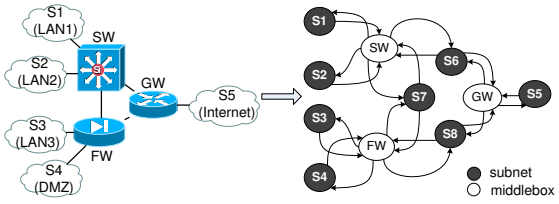

Fig. 2. Example network topology

Between any two directly connected middleboxes, our model assumes there is a subnetwork because the middlebox interfaces may have a distinct IP address and an ACL guarding that interface. The reachability of such a subnetwork is important because of several reasons. First, there are often some management services on a middlebox (such as SNMP, Telnet, and SSH) that are accessible through each interface. Such services are intended to be used only by administrators. Therefore, there are often some rules in the ACL deployed on each interface to restrict the access to this subnetwork. Second, if a middlebox is compromised, the source of the subsequent attacks is the IP address of an interface of the middle box; thus, the reachability from that subnetwork to other subnetworks is critical. Indeed, if the interfaces are not assigned IP addresses, the subnet is modeled with an empty address set; henceforth, reachability to and from the subnetwork is empty.

An ACL consists of a list of rules, where each rule has a predicate over some packet header fields and a decision (*i.e.*, action) to be taken for the packets that match the predicate. The decision of a rule is typically *accept* (*i.e.*, *permit*) or *discard* (*i.e.*, *deny*). As a packet may match two rules in an ACL and the two rules may have different decisions, the decision for a packet is the decision of the first (i.e., highest priority) rule that the packet matches. Table I shows an example ACL.

Rule	Src IP	Dst IP	Src Port	Dst Port	Proto.	Action
r_1	35.9.1.4/24	192.168.0.1	*	80	TCP	accept
r_2	*	*	*	*	*	discard

TABLE I
AN EXAMPLE ACL

Table II lists important notations used in this paper.

B. Reachability Formulation

Network reachability depends on not only some static factors, *i.e.*, topology and ACL configurations, but also some dynamic factors, *i.e.*, routing states, where each is defined as a snapshot of all the routing tables of the middleboxes in the network, and the one-to-one mapping tables of dynamic NATs and PATs. We formulate three types of network reachability: lower-bound reachability, instantaneous reachability, and upper-bound reachability for a given network topology and the ACL configurations. We define the *lower-bound reachability* from subnet N_i to N_j as the set of packets that can go from N_i to N_j *at any time*. We define the *instantaneous reachability* from subnet N_i to N_j as the set of packets that can go from N_i to N_j *at a particular time*. We define the *upper-bound reachability* from subnet N_i to N_j as the maximal set of packets that can go from N_i to N_j *at some time*.

Below we formulate instantaneous, upper-bound, and lower-bound reachability. In our notations, we use "I", "U", and "L" to denote instantaneous, upper-bound, and lower-bound reachability, respectively; we further use "CL" and "CO" to denote connectionless and connection-oriented protocols.

*1) **Instantaneous Reachability**:* Given a network routing state s, which is a snapshot of all the routing tables of the middleboxes in the network, let $P_{i,j}(s)$ denote the path from N_i to N_j at state s, and M denote the number of hops/middleboxes on path $P_{i,j}(s)$. For the k-th middlebox, we use C_{2k-1} to denote the ACL on the incoming middlebox interface and C_{2k} to denote the ACL on the outgoing middlebox interface.

Notation Table			
$\mathcal{N}$	set of subnets	n	# of subnets
$\mathcal{R}$	set of middleboxes	m	# of middleboxes
h	max # of unidirectional interfaces on a middlebox	$\mathcal{I}$	network incident matrix
		$\mathcal{A}$	ACL matrix
p	number of the paths in a network	s	routing state
$P_{i,j}(s)$	path from N_i to N_j at routing state s	M	# of hops on a path
		C_k	k-th classifier in a path
$A(C_k)$	packets accepted by classifier C_k		from N_i to N_j
z	max # of paths between any two subnets	R	reachability set
		ℓ	ACL
F_i	i-th field in FDD	F	FDD matrix
f	Firewall Decision Diagram (FDD)	P	path matrix
g	max # of rules in an ACL	N_i	subnet i
d	# of fields in each rule		

TABLE II
NOTATION TABLE

For connectionless protocols (mainly the UDP protocol), the instantaneous reachability from N_i to N_j is the intersection of the set of UDP packets accepted by every ACL on the path from N_i to N_j. Thus, we calculated instantaneous reachability as follows:

$$R^I_{CL}(i,j,s) = \bigcap_{k=1}^{2M} A_{UDP}(C_k) \quad (1)$$

where $A_{UDP}(C_k)$ is the set of UDP packets accepted by C_k.

For connection-oriented protocols (mainly the TCP protocol), the instantaneous reachability from N_i to N_j also depends on the reachability of the acknowledgment (ACK) messages from N_j to N_i. To incorporate the signaling path reachability of data path $P_{i,j}(s)$, we distinguish the statefulness of the intermediate middleboxes according to the following three cases: all middleboxes in $P_{i,j}(s)$ are stateful, all middleboxes in $P_{i,j}(s)$ are stateless, and $P_{i,j}(s)$ contains both stateful middleboxes and stateless middleboxes.

All middleboxes in $\mathbf{P_{i,j}(s)}$ are stateful: In any stateful middlebox on path $P_{i,j}(s)$, the state of every TCP session is stored in a state table to ensure that the corresponding signaling messages can traverse back from N_j to N_i. Such messages are not checked against any ACL of the middleboxes on path $P_{j,i}(s)$. When a signaling message does not match any entry in the state table of a stateful middlebox, the message is dropped and the connection will fail. Here we assume that the network is designed such that we have *path-coupled* signaling on stateful firewalls and NAT, which means that the forward data path and the backward signaling path contain the same set of middleboxes. The path-coupled property holds for most existing networks [9], [16]. Thus, when all middleboxes in $P_{i,j}(s)$ are stateful, the instantaneous reachability from N_i to N_j is the intersection of the set of TCP packets accepted by every ACL on the path from N_i to N_j. Therefore, we calculated instantaneous reachability for this case as follows, where $A_{TCP}(C_k)$ represents the set of TCP packets accepted by ACL C_k.

$$R^I_{CO}(i,j,s) = \bigcap_{k=1}^{2M} A_{TCP}(C_k) \quad (2)$$

All middleboxes in $\mathbf{P_{i,j}(s)}$ are stateless: If all the intermediate middleboxes are stateless, we not only need to consider the data path from N_i to N_j, but also the signaling path from N_j to N_i. Let $\tilde{A}$ represent the set of accepted packets where in each packet the values of source and destination IP address fields are swapped, and the values of source and destination port number fields are also swapped. Field swapping is needed because in one TCP session each data packet and its corresponding signaling packet have their IP addresses and port numbers in the opposite order. Note that when all middleboxes in $P_{i,j}(s)$ are stateless, we do not need path-coupled assumption. Thus, the instantaneous reachability for the connection-oriented and reliable protocols is the intersection of the set of accepted TCP packets in the data path and the set of accepted TCP packets in the signaling path. Therefore, we calculated instantaneous reachability for this case as follows, where the classifiers on path $P_{j,i}$ consists of $C'_1, C'_2, \ldots, C'_{2M'}$.

$$R^I_{CO}(i,j,s) = \bigcap_{k=1}^{2M} A_{TCP}(C_k) \bigcap_{k=1}^{2M'} \tilde{A}_{TCP}(C'_k) \quad (3)$$

(Here $\bigcap_{k=1}^{2M} A_{TCP}(C_k) \bigcap_{k=1}^{2M'} A_{TCP}(C'_k)$ denotes $A_{TCP}(C_1) \cap \cdots \cap A_{TCP}(C_{2M}) \cap A_{TCP}(C'_1)1 \cap \cdots \cap A_{TCP}(C'_{2M'})$.)

$\mathbf{P_{i,j}(s)}$ contains both stateful middleboxes and stateless middleboxes: For stateful middleboxes, we again need assumption of path-coupled signaling. For stateless middleboxes, we do not need this assumption. Thus, the instantaneous reachability on $P_{i,j}(s)$ is the intersection of the set of accepted packets of stateful middleboxes calculated by formula (2) and the set of accepted packets of stateless routers calculated by formula (3).

*2) **The Reachability Bounds**:* The *Reachability Lower-bound* from N_i to N_j, $R^L(i,j)$, denotes the set of packets that can traverse from N_i to N_j in *all* routing states. The *Reachability Upper-bound* from N_i to N_j, $R^U(i,j)$, denotes the maximal set of packets that can traverse from N_i to N_j in *some* routing states. Let $\mathcal{S}$ denote the set of all routing states of a network. The reachability lower-bound and upper-bound from N_i to N_j are calculated below:

$$R^U_{CL}(i,j) = \bigcup_{s\in\mathcal{S}} R^I_{CL}(i,j,s) \quad (4)$$

$$R^L_{CL}(i,j) = \bigcap_{s\in\mathcal{S}} R^I_{CL}(i,j,s) \quad (5)$$

Similar to the reachability bounds for connectionless protocols, the reachability bounds of the connection-oriented protocols using formulas (2) and (3) are calculated below:

$$R^U_{CO}(i,j) = \bigcup_{s\in\mathcal{S}} R^I_{CO}(i,j,s) \quad (6)$$

$$R^L_{CO}(i,j) = \bigcap_{s\in\mathcal{S}} R^I_{CO}(i,j,s) \quad (7)$$

Computing reachability lower-bound and upper-bound is very useful. For example, lower-bound reachability can be used to ensure that the available services on a subnet are reachable regardless of routing states, and upper-bound reachability can be used to ensure that the access to some services is restricted. Furthermore, the reachability upper-bound and lower-bound are useful in verifying the correctness of ACLs. Ideally, the reachability upper-bound and lower-bound from N_i to N_j should be the same (*i.e.*, $\Delta R(i,j) = R^U(i,j) - R^L(i,j)$ should be $\emptyset$). Otherwise, the ACLs have inconsistent decisions for the packets in $\Delta R(i,j)$: sometimes they are allowed to traverse from N_i to N_j, and sometimes they are not. For a packet $\pi \in \Delta R(i,j)$, if π should be constantly allowed to traverse from N_i to N_j, then blocking π at some routing states may disrupt legitimate services; if π should be constantly disallowed to traverse from N_i to N_j, then accepting π at some states may cause security breaches.

C. Reachability Formulation with NAT

Thus far, network reachability calculations are based on the assumption that packet header fields are not changed in the traversal from a source subnet to a destination subnet. Actually, there may be some packet transformers, such as Network Address Translation (NAT) and Port Address Translation (PAT), on the intermediate middleboxes that modify packet headers. A NAT transformer on a middlebox may change the source address field of a packet from x to x' and keep a record of this transformation in a table, which is used to change the destination field of the corresponding signaling packet from x' to x. A PAT transformer works similarly for port fields. Here, the path-coupled signaling assumption is necessary for paths that contain packet transforming filters.

Typically, a middlebox (such as a Cisco router [2] and IPtables [4]) applies NAT to a packet after it passes the ACL on the incoming interface and before it is sent to the ACL on the outgoing interface. Let middlebox γ be the one on path

$P_{i,j}(s)$ that uses a packet transformation (for source address or port number fields) function $T_S : N_i \mapsto N_i{}'$, where $N_i{}'$ is the virtual subnet to which N_i is mapped. We use T_S^{-1} to denote the reverse function. The instantaneous reachability for connectionless protocols is calculated using formula (1) as:

$$R^I_{CL}(i,j,s,T_S) = \bigcap_{k=1}^{2\gamma-1} A_{UDP}(C_k) \bigcap T_S^{-1}(\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)) \quad (8)$$

Note that applying function T_S^{-1} to $\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)$ means changing the source fields of every packet in $\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)$ from $N_i{}'$ to N_i.

The reachability bounds for connectionless protocols are calculated using formulas (4) and (5) as follows:

$$R^U_{CL}(i,j,T_S) = \bigcup_{s\in\mathcal{S}} R^I_{CL}(i,j,s,T_S) \quad (9)$$

$$R^L_{CL}(i,j,T_S) = \bigcap_{s\in\mathcal{S}} R^I_{CL}(i,j,s,T_S) \quad (10)$$

For connection-oriented protocols, the middlebox γ in the data path is the middlebox γ' in the signaling path (based on the path-coupled assumption). The instantaneous reachability formulation for data paths $R^{\rightarrow}_{CO}(i,j,s,T_S)$ is as follows:

$$R^{\rightarrow}_{CO}(i,j,s,T_S) = \bigcap_{k=1}^{2\gamma-1} A_{TCP}(C_k) \bigcap T_S^{-1}(\bigcap_{k=2\gamma}^{2M} A_{TCP}(C_k)) \quad (11)$$

Similarly, the instantaneous reachability formulation for signaling path $R^{\leftarrow}_{CO}(j,i,s,T_D)$ is as follows:

$$R^{\leftarrow}_{CO}(j,i,s,T_D) = T_D^{-1}(\bigcap_{k=1}^{2\gamma'-1} A_{TCP}(C'_k)) \bigcap_{k=2\gamma'}^{2M'} A_{TCP}(C'_k) \quad (12)$$

where T_D transforms the destination addresses of signaling packets from N_i to N_i'.

Using formulas (11), (12), and (3), we formulate instantaneous reachability for connection-oriented protocols as:

$$R^I_{CO}(i,j,s,T_S,T_D) = R^{\rightarrow}_{CO}(i,j,s,T_S) \cap \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \quad (13)$$

Note that formula (13) can be easily generalized to handle the paths that have multiple packet transformers.

The reachability bounds for connection-oriented protocols are formulated based on equations (6) and (7) as follows:

$$\begin{aligned} R^U_{CO}(i,j,T_S,T_D) &= \bigcup_{s\in\mathcal{S}} R^I_{CO}(i,j,s,T_S,T_D) \\ &= \bigcup_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,T_S) \bigcap \bigcup_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \end{aligned} \quad (14)$$

$$\begin{aligned} R^L_{CO}(i,j,T_S,T_D) &= \bigcap_{s\in\mathcal{S}} R^I_{CO}(i,j,s,T_S,T_D) \\ &= \bigcap_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,T_S) \bigcap_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \end{aligned} \quad (15)$$

III. Algorithms for Computing Reachability Matrices

In this section, we present algorithms for computing reachability for networks with no packet transformation filters. Reachability quantification algorithms for networks with packet transformation filters are described in the technical report version of this paper [8].

A. Reachability Matrices

We represent network reachability as the six matrices shown below. We use n to denote the number of subnets, and z to denote the maximum number of paths between any pair of subnets.

1) $A^I_{CL}[1..n,1..n,1..z]$ where each element $A^I_{CL}[i,j,k]$ is the set of packets representing the instantaneous reachability from N_i to N_j on the k-th path for connectionless protocols.
2) $A^I_{CO}[1..n,1..n,1..z,1..z]$ where each element $A^I_{CO}[i,j,k,\ k']$ is the set of packets representing the instantaneous reachability from N_i to N_j on the k-th data path and the k'-th signaling path for connection-oriented protocols.
3) $A^L_{CL}[1..n,1..n]$ where each element $A^L_{CL}[i,j]$ is the set of packets representing the lower-bound reachability from N_i to N_j for connectionless protocols.
4) $A^L_{CO}[1..n,1..n]$ where each element $A^L_{CO}[i,j]$ is the set of packets representing the lower-bound reachability from N_i to N_j for connection-oriented protocols.
5) $A^U_{CL}[1..n,1..n]$ where each element $A^U_{CL}[i,j]$ is the set of packets representing the upper-bound reachability from N_i to N_j for connectionless protocols.
6) $A^U_{CO}[1..n,1..n]$ where each element $A^U_{CO}[i,j]$ is the set of packets representing the upper-bound reachability from N_i to N_j for connection-oriented protocols.

B. Basic Data Structures and Algorithms

For any ACL ℓ, we define the *accept set* of ℓ, denoted $accept(\ell)$, to be the set of packets that can be accepted by ℓ. In this section, we first consider the following core problem in computing network reachability matrices: given two ACLs ℓ_1 and ℓ_2, how can we compute $accept(\ell_1) \cap accept(\ell_2)$ and $accept(\ell_1) \cup accept(\ell_2)$? Our algorithm for this computation consists of three steps: FDD construction, FDD shaping, and FDD logical operations.

1) Step I - FDD Construction: In this step, we convert each ACL to an equivalent Firewall Decision Diagram (FDD). FDD was introduced by Gouda and Liu in [5] as a data structure for representing access control lists. As network reachability only concerns whether a packet is accepted or discarded, in this paper, it is sufficient to use only the FDDs whose decisions are $\{1,0\}$ where 1 represents *accept* and 0 represents *discard*. We call such FDDs "Binary FDDs". In converting an ACL to an equivalent binary FDD, we replace all flavors of *accept*, such as *accept* and *accept with logging*, by 1, and replace all flavors of *discard*, such as *discard*, *reject*, and *discard/reject with logging*, by 0. We further define a *full-length ordered FDD* as an FDD where in each decision path, all fields appear exactly once and in the same order. For ease of presentation, in the rest of this paper, we use the term "FDD" to mean "binary full-length ordered FDD" if not otherwise specified. Figure 3(b) shows the two FDDs constructed from the two ACLs in 3(a).

An FDD construction algorithm, which converts a sequence of range rules to an equivalent full-length ordered FDD, is described in [11]. For computing reachability matrices, we choose the protocol type field as the label of the root node.

We call the decision paths whose terminal nodes are labeled 1 *accept paths*. Similarly, we call the decision paths whose terminal nodes are labeled 0 *discard paths*. Given an ACL ℓ,

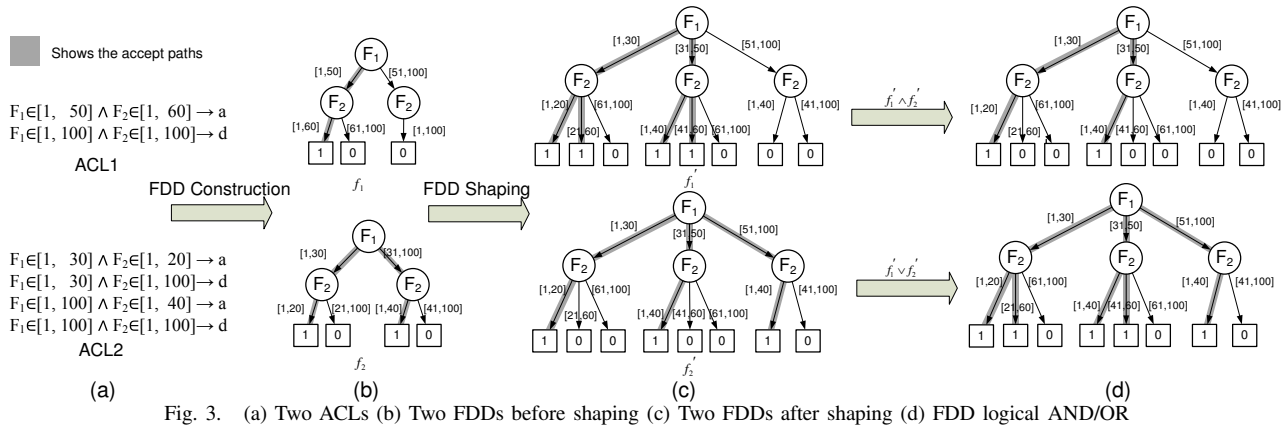

Fig. 3. (a) Two ACLs (b) Two FDDs before shaping (c) Two FDDs after shaping (d) FDD logical AND/OR

after we convert it to an equivalent FDD f, the accept paths of f represent the set $accept(\ell)$.

2) STEP II - FDD Shaping: In the previous step, we convert the two given ACLs ℓ_1 and ℓ_2 to two FDDs f_1 and f_2 such that ℓ_1 is equivalent to f_1 and ℓ_2 is equivalent to f_2. In this step, we further convert the two FDDs f_1 and f_2 to another two FDDs $f_1{}'$ and $f_2{}'$ such that the following three conditions hold: (1) f_1 is equivalent to $f_1{}'$, (2) f_2 is equivalent to $f_2{}'$, and (3) $f_1{}'$ and $f_2{}'$ are semi-isomorphic. Two FDDs are *semi-isomorphic* if and only if they are exactly the same except the labels of their terminal nodes [11]. The algorithm for equivalently converting two FDDs to two semi-isomorphic FDDs is described in [11]. Fig. 3(c) shows the two semi-isomorphic FDDs converted from the two FDDs in Fig. 3(b).

3) STEP III - FDD Logical AND/OR Operations: In previous steps, we equivalently convert two given ACLs ℓ_1 and ℓ_2 to two semi-isomorphic FDDs $f_1{}'$ and $f_2{}'$. In this step, we compute $accept(\ell_1) \cap accept(\ell_2)$ and $accept(\ell_1) \cup accept(\ell_2)$ using $f_1{}'$ and $f_2{}'$.

For any two semi-isomorphic FDDs $f_1{}'$ and $f_2{}'$, we define $f_1{}' \wedge f_2{}'$ as a new FDD f such that f is semi-isomorphic to $f_1{}'$ (and $f_2{}'$) and a terminal node in f is labeled 1 if and only if the two corresponding nodes in $f_1{}'$ and $f_2{}'$ are both labeled 1 (otherwise is labeled 0). This implies that the accept paths of $f_1{}' \wedge f_2{}'$ are the intersection of the set of accept paths in $f_1{}'$ and that in $f_2{}'$. Therefore, we can calculate $accept(\ell_1) \cap accept(\ell_2)$ by calculating the accept paths in $f_1{}' \wedge f_2{}'$.

Similarly, for any two semi-isomorphic FDDs $f_1{}'$ and $f_2{}'$, we define $f_1{}' \vee f_2{}'$ as a new FDD f such that f is semi-isomorphic to $f_1{}'$ (and $f_2{}'$) and a terminal node in f is labeled 0 if and only if the two corresponding nodes in $f_1{}'$ and $f_2{}'$ are both labeled 0 (otherwise is labeled 1). This implies that the accept paths of $f_1{}' \vee f_2{}'$ are the union of the set of accept paths in $f_1{}'$ and that in $f_2{}'$. Therefore, we can calculate $accept(\ell_1) \cup accept(\ell_2)$ by calculating the accept paths in $f_1{}' \vee f_2{}'$.

C. Computing Path and FDD Matrices

We next discuss the computing of two matrices, called path matrix and FDD matrix, which will be used in computing reachability matrices. The path matrix P is an $n \times n$ matrix where each element $P[i,j]$ is the *set* of one-way paths from N_i to N_j. For each path, letting $\ell_1, \cdots, \ell_h$ be the ACLs along the path, we compute the FDD that represents $accept(\ell_1) \cap \cdots \cap accept(\ell_h)$. The resulting FDDs are stored in the FDD matrix F, which is also an $n \times n$ matrix.

First, we initialize matrices P and F as follows. For any $1 \le i, j \le n$, if there is path from N_i to N_j via a middlebox, then $P[i,j]$ consists of this path (which is composed of two links: the link from N_i to the middlebox and the link from the middlebox to N_j) and $F[i,j]$ consists of the FDD that represent the intersection of the accept sets of the two ACLs associated with the two links; otherwise, $P[i,j]$ and $F[i,j]$ are both empty.

Second, we complete matrices P and F based on formulas (16) and (17) using dynamic programming. We use $P[i,k] \circ P[k,j]$ to denote the set of paths where each path is a concatenation of a path in $P[i,k]$ and a path in $P[k,j]$. Similarly, we use $F[i,k] \wedge F[k,j]$ to denote the set of FDDs where each FDD is the logical AND of an FDD in $F[i,k]$ and an FDD in $F[k,j]$. Note that we remove all paths with cycles because cycles are typically prevented by routing protocols.

$$P[i,j] = \bigcup_{k \in \mathcal{N}} P[i,k] \circ P[k,j] \tag{16}$$

$$F[i,j] = \bigcup_{k \in \mathcal{N}} F[i,k] \wedge F[k,j] \tag{17}$$

Third, for each FDD in $F[i,j]$, we reduce the domain of the source IP address field to the set of IP addresses used in subnet N_i and the domain of the destination IP address field to the set of IP addresses used in N_j. We use $src(N_i)$ to denote the set of packets whose source IP address is in N_i and $dst(N_j)$ to denote the set of packets whose destination IP address is in N_j. We use $fdd(src(N_i))$ to denote the FDD that represents $src(N_i)$ and $fdd(src(N_j))$ to denote the FDD that represents $dst(N_j)$. Therefore, in this step, we replace each FDD f in $F[i,j]$ by $fdd(src(N_i)) \wedge f \wedge fdd(dst(N_j)$.

D. Computing Reachability Matrices

We are now ready to compute the 6 reachability matrices.

1) Reachability for Connectionless Protocols: For any $1 \le k \le |F[i,j]|$, we use $F[i,j]_k$ to denote the k-th FDD in $F[i,j]$ and $P[i,j]_k$ to denote the k-th path in $P[i,j]$. We use $Sub_{CL}(F[i,j]_k)$ to denote the UDP subtree of FDD $F[i,j]_k$. Recall that in computing reachability we choose protocol type to be the label of the root node. Therefore, the instantaneous reachability of the path $P[i,j]_k$ is:

$$A^I_{CL}[i,j,k] = Sub_{CL}(F[i,j]_k) \tag{18}$$

Accordingly, based on formulas (4) and (5), the reachability upper-bound and lower-bound from N_i to N_j are calculated as follows:

$$A^U_{CL}[i,j] = Sub_{CL}(\bigvee_{k=1}^{|F[i,j]|} F[i,j]_k) \tag{19}$$

$$A^L_{CL}[i,j] = Sub_{CL}(\bigwedge_{k=1}^{|F[i,j]|} F[i,j]_k) \quad (20)$$

2) Reachability for Connection-oriented Protocols: We first consider the case that all middleboxes on paths from N_i to N_j are stateful. We use $Sub_{CO}(F[i,j]_k)$ to denote the TCP/ICMP subtree. The instantaneous, upper-bound, and lower-bound reachability matrices are calculated using formulas (2), (6) and (7), as follows:

$$A^I_{CO}[i,j,k] = Sub_{CO}(F[i,j]_k) \quad (21)$$

$$A^U_{CO}[i,j] = Sub_{CO}(\bigvee_{k=1}^{|F[i,j]|} F[i,j]_k) \quad (22)$$

$$A^L_{CO}[i,j] = Sub_{CO}(\bigwedge_{k=1}^{|F[i,j]|} F[i,j]_k) \quad (23)$$

Second, we consider the case that all middleboxes on paths from N_i to N_j are stateless. As discussed in Section II-B1, for the instantaneous reachability, we need to look at the reachability of each data path $P[i,j]_k$ and the corresponding signaling path $P[j,i]_{k'}$. The swapping operator is implemented by function $Swap_{SD}$. For an FDD f, the function $Swap_{SD}(f)$ basically swaps the labels of source fields and destination fields. The instantaneous, upper-bound, and lower-bound reachability matrices are calculated using formulas (3), (6), and (7), as follows:

$$A^I_{CO}[i,j,k,k'] = Sub_{CO}(F[i,j]_k \wedge Swap_{SD}(F[j,i]_{k'})) \quad (24)$$

$$A^U_{CO}[i,j] = \bigvee_{k=1}^{|F[i,j]|} \bigvee_{k'=1}^{|F[j,i]|} A^I_{CO}[i,j,k,k'] \quad (25)$$

$$A^L_{CO}[i,j] = \bigwedge_{k=1}^{|F[i,j]|} \bigwedge_{k'=1}^{|F[j,i]|} A^I_{CO}[i,j,k,k'] \quad (26)$$

For the case that the paths from N_i to N_j contain both stateful and stateless middleboxes, we use the formulas (21), (22), and (23) to handle the stateful middleboxes and formulas (24), (25), and (26) to handle stateless middleboxes.

E. Complexity Analysis

For a given network, let n be the number of subnets, m be the number of middleboxes, h be the maximum number of interfaces on a middlebox, p be the number of the paths in the network, g be the maximum of number of rules in an ACL, and d be the number of fields in each rule. Note that d is typically a constant, which is 4 or 5 for IP networks.

The complexity of constructing the equivalent FDD from an ACL with g d-dimensional rules is $O(g^d)$ [11]. The complexity of constructing FDDs from all ACLs is $O(g^d \cdot h \cdot m)$. The complexity of shaping the two FDDs constructed from two ACLs is $O((2g)^d) = O(g^d)$. Therefore, the complexity of computing reachability matrices is $O(p \cdot g^d)$.

In theory, the total number of paths p is exponential in terms of the number of subnets and middleboxes in the network. However, in practice, p is much smaller than its theoretical upper-bound because networks are typically designed following the hierarchical network design model [1]. Using this model, a network is designed in three layers, namely a core layer, a distribution layer, and an access layer. Security policies are mostly applied on the distribution layer, and the core layer is mainly used to facilitate routing between subnets. For networks designed by this model, the number of paths between two subnets is typically small (often one), and the length of a path is typically small.

IV. Online Reachability Queries

After reachability matrices are calculated, we can use them as the engine for efficiently processing network reachability queries. In this section, we discuss languages for specifying reachability queries, ways of using such queries for network and security management, and algorithms for processing these queries. Based on the nature of queries, Quarnet supports three types of queries: upper-bound, lower-bound, and instantaneous. Upper-bound/lower-bound reachability queries are useful in verifying whether the ACLs on middleboxes satisfy certain security policies. Instantaneous reachability queries are useful for real-time security monitoring as the administrator identifies which paths are used at the time of querying. Such queries are also useful to verify whether the changes on the ACLs on some middleboxes have undesired impact on reachability. Based on the answer of queries, Quarnet supports two types of queries: *closed* and *open*. A closed query demands an answer of *yes/no*. For instance, considering the network in Figure 2, can all hosts in S1 communicate with Mail Server in S4 on TCP port 25 via any path? An open query demands an answer in terms of a *set*. For example, which hosts in S1 can access the Mail Server in S4 on TCP port 25 via any path from S1 to S4? As another example, what set of paths may let all hosts in S1 access the Mail Server in S4 on TCP port 25?

A. Reachability Query Language

SRQL Syntax: We define an SQL-like language called *Structured Reachability Query Language* (SRQL) for specifying reachability queries. SRQL has the following format:

```
reachability_type 𝒯
connection_type 𝒪
select ℱ
where (F₁ ∈ S₁) ∧ ··· ∧ (F_d ∈ S_d) ∧ (𝒫 ∈ S_P)
```

The reachability type $\mathcal{T}$ denotes the type of reachability, namely instantaneous (I), upper-bound (U), or lower-bound (L). The connection type $\mathcal{O}$ denotes the connection orientation of transport protocols, namely connection-oriented (CO) or connectionless (CL). When the reachability type $\mathcal{T}$ is upper-bound or lower-bound, the `select` clause $\mathcal{F}$ is a subset of packet fields $\{F_1, F_2, \cdots, F_d\}$; when $\mathcal{T}$ is instantaneous, $\mathcal{F}$ is a subset of fields $\{F_1, F_2, \cdots, F_d, \mathcal{P}\}$ where $\mathcal{P}$ denotes the attribute of "path". In the `where` clause, the predicate $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d)$ specifies the set of packets that this query is concerned with and $(\mathcal{P} \in S_P)$ specifies the set of paths that this query concerns. For example, SRQL query for the question "Through what paths the mail server in S4 on TCP port 25 is accessible from S1?" is the following:

```
type 𝒯
protocol CO
select 𝒫
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

Note that we do not expect administrators to specify queries using SRQL directly. Instead, we expect a full-fledged implementation of Quarnet to provide a GUI interface for inputting queries and specifying paths. The SRQL will be used to formally represent a query under the hood.

SRQL Semantics: The result of an upper-bound reachability query, where $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_h\}$ and $\mathcal{F}_i \in \{F_1, F_2, \cdots, F_d\}$ for every $1 \leq i \leq h$, is defined as follows:

$\{(\pi_{\mathcal{F}_1}, \cdots, \pi_{\mathcal{F}_h}) | (\pi_1 \in S_1) \wedge \cdots \wedge (\pi_d \in S_d)$ and packet $(\pi_1, \cdots, \pi_d)$ can traverse from its source to its destination at some time$\}$

The result of a lower-bound query is defined similarly except that "at some time" is replaced by "at any time".

The result of an instantaneous reachability query, where $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_h, \mathcal{P}\}$ and $\mathcal{F}_i \in \{F_1, F_2, \cdots, F_d\}$ for every $1 \leq i \leq h$, is defined as follows:

$\{(\pi_{\mathcal{F}_1}, \cdots, \pi_{\mathcal{F}_h}, \rho) | (\pi_1 \in S_1) \wedge \cdots \wedge (\pi_d \in S_d)$ and packet $(\pi_1, \cdots, \pi_d)$ can traverse from its source to its destination through path ρ where $\rho \in S_P.\}$

SRQL Example 1: We next give some query examples, where we use the shorthand S for source IP, D for destination IP, SP for source port, DP for destination port, and PT for protocol type. The question "Can all hosts in S1 communicate with the mail server in S4 on TCP port 25?" can be formulated as the following query:

```
type L
protocol CO
select S
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

If the query result is all the IP addresses in S1, then the answer is "yes"; otherwise the answer is "no".

SRQL Example 2: SRQL query for the question "Through what paths the mail server in S4 on TCP port 25 is accessible from S1?" is the following:

```
type I
protocol CO
select P
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

B. Reachability Query Engine Construction

Our reachability query engine consists of six FDDs representing the six reachability matrices. We compute the six FDDs as follows. For each of the four upper-bound/lower-bound reachability matrices, we apply the logical OR operation to all matrix elements, where each element is an FDD representing the reachability between two specific subnets. The resultant FDD over d fields represents the upper-bound/lower-bound reachability between any two subnets. For each of the two instantaneous reachability matrices, we compute the two corresponding FDDs as follows. First, we reduce the two instantaneous reachability matrices to 2-dimensional matrices by combining the FDDs for the various paths from a source to a destination into one FDD. To achieve this, we first add a new node labeled with a new attribute "*path*" to each FDD as the root whose outgoing edge is labeled with path IDs, and then apply the logical OR operation to all FDDs regarding the reachability from one subnet to another. It is trivial to label every path with a unique ID. Second, for each of the two resultant 2-dimensional matrices, we apply the logical OR operation to all elements and get an FDD over d+1 fields. The six FDDs will be used to process SRQL queries.

C. Online Reachability Query Processing

Reachability queries can be quickly processed by traversing one of the six FDDs computed above. The algorithm is essentially the same as the one described in [10] for querying one firewall policy.

V. Experimental Results

A. Reachability Computation

We implemented Quarnet in C++ and evaluated it on a university campus network. We focused the measurement on the execution time and memory usage of Quarnet. We concluded that Quarnet is sufficiently efficient to be used in practice as off-line computation, although computing network reachability is a resource consuming task in nature.

This campus network consists of 48 subnets interconnected in a hierarchical topology. We model the core campus network as one subnet as most ACLs are deployed on edge routers/firewalls. We disregard the ACLs deployed on the core routers in the network because we have no access to them. Furthermore, their potential impact on the reachability calculation is expected to be small because the ACLs on core routers are typically very small and they mostly specify a few rules allowing only administrators to access the middleboxes for management purposes [1]. One subnet may contain multiple Virtual Local Area Networks (VLANs) and there are no ACLs among VLANs. For example, a VLAN could be the network that consists of all the printers in the subnet of a department. Note that we model the outside Internet as one subnet. This network does not have NATs/PATs. Further, all ACLs that we obtained are in use on stateful firewalls. In this network model, there are 49 subnets that are connected by 192 links through 2401 paths. Also, the total number of VLANs is 399 that are protected by 98 ACLs, where each ACL contains 573 rules in average (total number of rules: 56189). Among the 98 ACLs, 14 of them are original and the rest are generated based on the statistical features of the original ones and the subnet addresses because we do not have access to all the ACLs in deployment.

We conducted experiments on a desktop computer with a Dual Core AMD64 CPU 2.4GHz and 16GB of RAM. This is a public machine running processes from other users as well. Our experimental results are shown in Table III. Note that the total amount of memory used by Quarnet is 4.7GB, not the sum of the memory used in each step because some memory is released after each intermediate step.

	# of FDDs	Time (mins)	RAM (MB)
FDD construction	192	2.1	1276
One-hop path calculation	96	0.3	106
FDD matrix calculation	2352	11	1505
Reachability matrices calculation for connection-less protocols	2352	661	2400
Reachability matrices calculation for connection-oriented protocols	2352	1	800
Total execution	7344	675	4700

TABLE III
RESULTS ON COMPUTING REACHABILITY MATRICES

We gained two insights from our experiments. First, the running time of our algorithm goes up as the number of VLANs increases. This is because more number of VLANs means more intervals on the outgoing edges of nodes labeled with source or destination fields, which further means more edge splitting and subtree copying in performing FDD shaping. Second, the time for computing reachability matrices for connection-oriented protocols goes down dramatically as the number of stateful firewalls increases. This is because based on formulas (21), (23), and (22), we can use the FDDs calculated in the connectionless matrices and simply change the subtree functions from Sub_{CL} to Sub_{CO}.

B. Experimental Results Validation

To validate the reachability matrices computed by Quarnet, we designed an ACL simulator that makes the decision for

each packet by sequentially comparing the packet with every rule in an ACL. We generated a large number of packets for every path including all corner case packets based on the bounding values in the ACLs. We compared the decisions made by the ACL simulator and those made by the reachability matrices computed by Quarnet. The results are all positive.

C. Performance of Core Operations

The core operations in reachability computation are FDD logical AND and OR operations, the performance of which is the same as they all come down to FDD shaping. We are interested in evaluating their performance because it helps us to estimate the running time and memory usage of reachability matrix calculation. To evaluate the performance of the core operations, we created synthetic rules generated based on the statistical features of the real rules that we have obtained, such as the probability of unique IP addresses and port numbers, the decision bias of the rules, and the probability of fields being `any`. We focused on measuring the time and memory for calculating the reachability of paths with different lengths.

Figure 4 (a) and (b) show the running time and memory usage of the core operations over different path lengths for a network with 100 subnets. The average number of rules per ACL ranges from 100 to 400 with 10% STD based on normal distribution. Note that based on the dynamic programming in Section III-C, the FDD for each path is calculated using a single logical operation. For instance, the FDD for a path of length 8 is calculated by an AND operation of either two FDDs where each is for a path of length 4 or two FDDs, one for a path of length 6 and one for a path of length 2. For any path length, we calculate the average cost of all possible permutations by which the FDD of the path may be calculated.

The interesting observation on Figure 4 is that in general the running time and memory usage decrease as the path length increases. This makes sense because the rules in different ACLs from one network often share common fields. The source and destination IP address fields of many rules in the ACLs from one network are drawn from the IP prefixes of the subnets in the network. Similarly, many of the port number and protocol fields are drawn from the set of services provided by the network. When we shape two FDDs each corresponds to a long path, because each of the two FDDs has gone through many edge splitting, the amount of new edge splitting tends to be small, which leads to reduced cost.

Knowing the performance of Quarnet core operations allows us to estimate the time and memory that it used to compute the reachability matrices for a given network. Let $\mathcal{C}(x)$ be the cost function for calculating the FDD for a path with length x as shown in figure 4. Let $\mathcal{L}(P[i,j]_k)$ be the length of the path $P[i,j]_k$. We use $\mathbb{C}^I_{CL}$, $\mathbb{C}^I_{CO}$, $\mathbb{C}^B_{CL}$, and $\mathbb{C}^B_{CO}$ to denote the estimated cost (*i.e.*, running time or memory usage) for computing

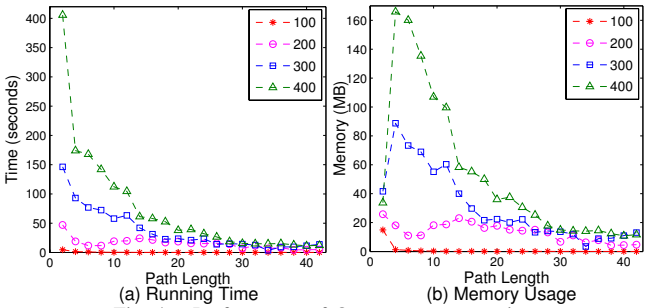

Fig. 4. Performance of Quarnet core operations

the matrices of instantaneous reachability for connectionless protocols, instantaneous reachability for connection-oriented protocols, reachability bounds for connectionless protocols, and reachability bounds for connection-oriented protocols, respectively. They can be calculated as follows:

$$\mathbb{C}^I_{CL} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\mathcal{C}(\mathcal{L}(P[i,j]_k))$$

$$\mathbb{C}^I_{CO} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\sum_{k'=1}^{|P[j,i]|}\mathcal{C}(\mathcal{L}(P[i,j]_k)+\mathcal{L}(P[j,i]_{k'})))$$

$$\mathbb{C}^B_{CL} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=2}^{|P[i,j]|}\mathcal{C}(\sum_{k'=1}^{k}\mathcal{L}(P[i,j]_{k'}))$$

$$\mathbb{C}^B_{CO} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\sum_{l=1}^{|P[j,i]|}\mathcal{C}(\sum_{k'=1}^{k}\mathcal{L}(P[i,j]_{k'})+\sum_{l'=1}^{l}\mathcal{L}(P[j,i]_{l'}))$$

D. Performance of Online Querying

As our reachability query engine uses FDDs as its core data structures, we evaluated the performance of online query processing by performing randomly generated queries over large FDDs with millions of nodes. Our experimental results in Table IV show that our online reachability query engine is very efficient. For example, over an FDD with 2 million nodes, which is similar to the size of the FDDs uses in the online query engine built for the university campus network that we experimented, the average processing time for a query is 0.075 seconds, although some queries (less than 1%) take 2-3 seconds. The existence of some outliers is because some randomly generated queries may cause the engine to search through a large portion of the FDD.

FDD Size (# nodes)	Average Query Processing Time	Outliers
0.5 million	0.032s	1% of queries: 0.5s $\sim$ 1s
1 million	0.049s	0.8% of queries: 0.8s $\sim$ 1.5s
2 million	0.075s	1% of queries: 2s $\sim$ 3s

TABLE IV
PERFORMANCE OF ONLINE QUERY PROCESSING

VI. Related Work

Active probing tools, which actively test network reachability by sending probing packets and analyzing the response packets, are commonly used by network administrators. Such tools include `ping` and `traceroute`, which use ICMP echo request/reply or ICMP time-exceed packets to test whether a host on target network is reachable. The use of such tools is limited because they cannot verify the reachability of UDP or TCP packets. There are tools such as NMAP and NESSUS that can test the reachability of UDP or TCP packets; however, such tools have significant limitations. First, they cannot perform comprehensive testing due to the amount of packets that have to be generated. Second, the test results of such tools are valid only for the routing state at the time that the testing is performed and may not hold afterwards due to the change of routing states over time. Third, such tools only show the open ports on which a server daemon is listening and does not reveal the open ports with no server listening on them at the time of testing. In comparison, our Quarnet is non-intrusive and comprehensive. But of course, active probing tools have some benefits that Quarnet does not offer. For example, they may identify reachability faults, such as errors in routing software, which are not caused by ACL misconfigurations. Nevertheless, our tool is complementary to such tools.

Little work has been done on network reachability analysis. Xie *et al.* presented a model of network reachability in their seminal work [19]; however, they give no algorithms for computing reachability (and of course no experimental results). Xie *et al.*'s network reachability model does not address dynamic NAT (Network Address Translation) and PAT (Port Address Translation), and does not take into account whether transport layer protocols are connectionless or connection-oriented. Furthermore, Xie *et al.*'s model is limited to describing the networks where each subnet connects to only one router because they model a network as a graph over only routers. We significantly go beyond Xie *et al.*'s work along three dimensions: (1) reachability modeling and formulation, (2) algorithms for computing reachability, (3) solutions for reachability queries. Our model differs from Xie *et al.*'s work in the following aspects. First, we model a network as a graph over both routers and subnets, while Xie *et al.* model a network as a graph over only routers. Thus, Xie *et al.*'s model is limited to describing the networks where each subnet connects to only one router, while our model does not have this limitation. Furthermore, we calculate reachability between two subnets and they calculate reachability between two middleboxes. Second, we distinguish network reachability formulations based on both the properties of transport layer protocols (namely connectionless and connection-oriented protocols) and the statefulness of routers/firewalls on every path, while Xie *et al.* did not. Third, our model addresses three types of packet transformations: static NAT, dynamic NAT, and PAT, while Xie *et al.*'s model addresses only static NAT. Xie *et al.* gave no implementable algorithms for computing reachability and no solutions for reachability queries. Recently, in a poster paper [20], Zhang *et al.* proposed monitoring and verifying reachability in real-time by computing instantaneous reachability. However, they provided no algorithm for computing the reachability from a source to a destination along a given path and no experimental results. Furthermore, they have the other limitations of Xie *et al.*'s work mentioned above.

Some effort has been made to use Binary Decision Diagrams (BDDs) for reachability analysis [3], [17]; however, such work can only answer host-to-host queries and cannot answer subnet-to-subnet queries as we do in this paper. We choose FDDs instead of BDDs, as the basic data structure in this paper for several reasons. First, it is extremely difficult, if not infeasible, to swap the values of packet fields or rearrange packet fields in a BDD calculated over multiple ACLs. These operations are critical for computing reachability for connection-oriented protocols and networks with NAT/PAT. Note that our methods for performing these operations on FDDs, which requires generating rules from FDDs and reconstructing FDDs, cannot be applied to BDDs because generating rules from a BDD could easily lead to millions of rules as reported in [11]. Second, a reachability query engine built with BDDs can only process closed queries that demand a *yes*/*no* answer. In contrast, our reachability query engine built with FDDs can process both closed queries and open queries. Third, the reachability calculated by BDDs is not human readable. In contrast, every element in our reachability matrix is an FDD, which can be visualized for examination.

In [13], Mayer *et al.* proposed Fang, a firewall analysis engine. Fang supports limited queries (over 3-tuples) and each query is compared with every rule in every ACL along every path from a source to a destination, which is very inefficient.

There is some work, which is orthogonal to ours, on detecting reachability problems caused by routing faults (*e.g.*, the faults identified in [15]), instead of ACL misconfiguration (*e.g.*, [12]). In [6], Ingols *et al.* proposed algorithms for creating attack graphs for a network; however, their focus is not on reachability computation. Our proposed work is complementary to [6].

VII. Conclusions

We make four major contributions in this paper. First, we model and formulate network reachability considering the differences in connectionless and connection-oriented transport protocols, stateless and stateful middleboxes, as well as the presence and absence of various packet transformers. Second, we present algorithms for computing network reachability matrices. Third, we give solutions for expressing and processing reachability queries. Fourth, we implemented our algorithms and conducted experiments on a campus network. Results show that our offline reachability computation is practical and online query processing is very efficient.

References

[1] Internetworking design basics. *http://www.cisco.com/en/US/docs/internetworking/design/guide/nd2002.html*.

[2] NAT order of operation. *http://www.cisco.com/warp/public/556/5.html*, 2005.

[3] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *IEEE ICNP*, 2009.

[4] O. Andreasson. IPtables tutorial 1.2.2. *http://iptables-tutorial.frozentux.net/iptables-tutorial.html*, 2006.

[5] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.

[6] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *Proc. 22nd IEEE Annual Computer Security Applications Conference (ACSAC)*, pages 121–130, 2006.

[7] Z. Kerravala. As the value of enterprise networks escalates, so does the need for configuration management. Enterprise Computing & Networking, The Yankee Group Report, January 2004.

[8] A. Khakpour and A. X. Liu. Quarnet: A tool for quantifying static network reachability. Technical Report MSU-CSE-09-2, Michigan State Univesity, Dept. of Computer Science and Engineering, January 2009.

[9] A. Klenk, P. Schlicker, R. Kuhne, A. Fessi, C. Fan, F. Dressler, and G. Carle. Path coupled accounting mechanisms for all IP networks. In *6th IEE Int.Conf. on 3G & Beyond (3G 2005)*, 2005.

[10] A. X. Liu and M. G.Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(6):766–777, 2009.

[11] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(8), 2008.

[12] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proc. SIGCOMM*, 2002.

[13] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[14] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[15] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5:601–615, 1997.

[16] M. Stiemerling, H. Tschofenig, and C. Aoun. Nat/firewall nsis signaling layer protocol (nslp). *draft-ietf-nsis-nslp-natfw-05*, 2005.

[17] Y.-W. E. Sung, C. Lund, M. Lyn, S. G. Rao, and S. Sen. Modeling and understanding end-to-end class of service policies in operational networks. In *ACM SIGCOMM*, 2009.

[18] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

[19] G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. *Proc. IEEE INFOCOM*, 3:2170–2183, March 2005.

[20] B. Zhang, T. S. E. Ng, and G. Wang. Reachability monitoring and verification in enterprise networks. In *Proc. SIGCOMM*, 2008.