Patterns and Interactions in Network Security

PAMELA ZAVE and JENNIFER REXFORD, Princeton University

Networks play a central role in cyber-security: networks deliver security attacks, suffer from them, defend against them, and sometimes even cause them. This article is a concise tutorial on the large subject of networks and security, written for all those interested in networking, whether their specialty is security or not. To achieve this goal, we derive our focus and organization from two perspectives. The first perspective is that, although mechanisms for network security are extremely diverse, they are all instances of a few patterns. Consequently, after a pragmatic classification of security attacks, the main sections of the tutorial cover the four patterns for providing network security, of which the familiar three are cryptographic protocols, packet filtering, and dynamic resource allocation. Although cryptographic protocols hide the data contents of packets, they cannot hide packet headers. When users need to hide packet headers from adversaries, which may include the network from which they are receiving service, they must resort to the pattern of compound sessions and overlays. The second perspective comes from the observation that security mechanisms interact in important ways, with each other and with other aspects of networking, so each pattern includes a discussion of its interactions.

CCS Concepts: • **Networks** → *Network design principles*; *Network protocol design*; **Network security**; • **Security and privacy** → **Network security**; **Cryptography**; *Formal methods and theory of security*; **Security services**; *Systems security*; **Intrusion detection systems**;

Additional Key Words and Phrases: Network security

ACM Reference format:
Pamela Zave and Jennifer Rexford. 2020. Patterns and Interactions in Network Security. *ACM Comput. Surv.* 53, 6, Article 118 (December 2020), 37 pages.
https://doi.org/10.1145/3417988

1 INTRODUCTION

Today's Internet is not worthy of the trust society increasingly places in it. We hear every day about new security vulnerabilities and successful attacks, ranging from email viruses and Websites overrun with unwanted traffic, to network outages, compromised user data, and downright espionage. These attacks are costly, leading to denial of service, loss of revenue, identity theft, ransom demands, subversion of the democratic process, malfunctioning safety-critical equipment, and more.

Many successful security attacks use "social engineering" to prey on naive users, for example by getting them to click on malicious hyperlinks. Users are often guilty of using easily guessed passwords or failing to reset a default password on a new device. Application software is also the

Authors' address: P. Zave and J. Rexford, Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544; emails: pamela@pamelazave.com, jrex@cs.princeton.edu.

0360-0300/2020/12-ART118 $15.00
https://doi.org/10.1145/3417988

source of many security vulnerabilities, due to bugs or poor programming practices. Its complexity provides a big "attack surface" for adversaries to probe for weaknesses.

Despite the prevalence of social engineering and vulnerable applications, networks are an important part of the security landscape. Networks make attacks on applications possible by delivering unwanted traffic or leaking sensitive data. Network components and network services are often the targets of attacks. Sometimes a network itself is the adversary, performing unethical surveillance, or censoring communication.

Fortunately, networks can also be part of the solution, by blocking unwanted traffic, enabling anonymous communication, circumventing censorship, or protecting both infrastructure and applications from a range of known attacks. And network protocols can protect users by authenticating and encrypting communications.

This article is intended as a concise tutorial on the very large subject of security by and for networks, specifically the mechanisms through which network security is achieved. It is intended to be useful to all readers interested in networks, whether their specialty is security or not. Because the basic mechanisms have proven to be fairly stable over time, we do not emphasize which particular attacks and defenses are trending at the moment. The details of well-motivated attacks or cost-effective defenses change as technology changes, and particular defenses might cycle in and out of fashion. Instead, to achieve the goal of this article, we derive our focus and organization from two perspectives.

The first perspective is that, although mechanisms for network security are extremely diverse, they are all instances of just a few patterns. By emphasizing the patterns, we are able to cover more ground. We also aim to help the reader understand the big issues and retain the most important facts. The second perspective comes from the observation that security mechanisms interact in important ways, with each other and with other aspects of networking. These interactions deserve our attention. To provide communication services that are secure and also fully supportive of distributed applications, network designers must understand the consequences of their decisions on all aspects of network architecture and services.

The boundaries of network security have been drawn by convention over time, so we begin the tutorial by defining network security in two ways. First, there is a practical classification of network security attacks, based primarily on which agents are the attackers, defenders, and potential victims. The classification is based secondarily on defense mechanisms. Second, we discuss how network security is related to information security and other forms of cyber-security, as well as the gaps where no comprehensive defenses yet exist.

The four main sections of the tutorial cover the four major patterns for providing network security. All agents can protect their own communications with *cryptographic protocols* (Section 4), which (among other benefits) hide the data contents of packets. Networks can protect themselves and their users by *traffic filtering* (Section 5). Both users and networks can employ *dynamic resource allocation* to overcome attacks (Section 6). Although cryptographic protocols hide the data contents of packets, they cannot hide packet headers, because the network needs them to deliver the packets. So when users need to hide packet headers from adversaries, which may include their own network, they must resort to *compound sessions and overlays* (Section 7). The first three patterns will be familiar to anyone who has even dabbled in network security, while the importance of the fourth pattern has not been sufficiently recognized.

Between the definition of network security and the four major sections, Section 3 presents a new descriptive model of networks and network services. This model explains how network services are provided by means of composition of many networks at many levels of abstraction, where each network is self-contained in the sense of having—at least potentially—all the basic mechanisms of networking (such as routing, forwarding, session protocols, and directories). This model

	ATTACKERS	DEFENDERS AND DEFENSE MECHANISMS	POTENTIAL VICTIMS
FLOODING ATTACKS	unsafe users	network defends with traffic filtering and/or dynamic resource allocation safe users defend with dynamic resource allocation	network and/or safe users
SUBVERSION ATTACKS	unsafe users	network defends with traffic filtering and/or cryptographic protocols safe users defend with cryptographic protocols	network and/or safe users
POLICY VIOLATIONS	unsafe users	network defends with traffic filtering	authority responsible for the policy
SPYING AND TAMPERING	network and/or unsafe users	safe users defend with cryptographic protocols and/or compound sessions and overlays	safe users

Fig. 1. A practical classification of network security attacks.

allows complete and precise descriptions of today's network architectures. It is also necessary for recognition of the four patterns, because the same patterns are reused in different networks in a compositional architecture. The patterns are reusable precisely because the different networks have fundamental similarities, despite the fact that they may have different purposes, levels of abstraction, membership scope, or geographical span.

In each of the four main sections, in addition to presenting a security mechanism, we consider how the mechanism interacts with other mechanisms within its network and across composed networks. This helps to determine where security could and should be placed in a compositional network architecture.

A version of this tutorial with more details, examples, commentary, and references can be found on *arXiv* [41].

2 WHAT IS NETWORK SECURITY?

Network security is a pragmatic subject with boundaries that have been drawn by convention over time. Although the focus of this tutorial is defense mechanisms, we must have some idea of what kind of attacks they can defend against.

Classifying security attacks is extremely difficult because—by their very nature—security attacks are clever, they exploit gaps in standard models, and they are always evolving. In Section 2.1, we present a practical classification scheme based on multiple factors. It only covers known attacks, and there are some overlaps in the categories, but it does provide intuition that will be helpful for understanding the defenses.

Of all the factors relevant to security attacks, the worst factor for purposes of classification is real-world consequences (or, alternatively, the motivations of attackers). These consequences include financial loss, loss of time, loss of privacy, loss of reputation, loss of political freedom, loss of physical safety, and so on. Often, these losses are intertwined, because one loss causes another. Some attacks have no direct real-world consequences: their sole purpose is to enable other, more damaging, attacks.

Our practical classification scheme, summarized in Figure 1, is based primarily on which agents are the attackers, defenders, and potential victims. With one exception (see table in Figure 1), agents can be either *the network*, meaning the infrastructure machines provided by the network operator to run the network; *safe users,* meaning machines that use a network for communication and whose behavior is satisfactory according to whatever rules or authorities apply, or *unsafe users,* meaning machines that have access to a network and whose behavior is unsatisfactory because they have been programmed maliciously, ignorantly, or erroneously. Classification is based secondarily on defense mechanisms; these must be secondary to defenders because some mechanisms are only available to some defenders.

Note that the network is usually a defender, but can be an attacker. Even though traffic filtering is a possible defense for three attack categories, as we will explain below, the details of filtering against different attacks are quite different.

In Section 2.2, we will discuss alternative definitions of security. These include other kinds of cyber-security that complement network security, attacks for which comprehensive defenses do not yet exist, and alternative classification schemes.

2.1 A Practical Classification of Network Security Attacks

2.1.1 Flooding Attacks. In a *flooding attack*, attackers send floods of packets toward the victim, seeking to make it unavailable by exhausting its resources. Consequently, flooding attacks are one type of *denial-of-service attack* (see Section 2.2.1). The intended victims of flooding attacks vary. If the victim is a public server or user machine, the attack might seek to exhaust its compute-cycle, memory, or bandwidth resources. Note that some public servers such as DNS servers are part of the infrastructure of a network, so a flooding attack on a DNS server is an attempt to deny some network services to a large number of users. An attacker might also target some portion of a network, seeking to exhaust the bandwidth of its links. A bandwidth attack can make particular users unreachable and can also deny network service to many other users whose packets pass through the congested portion of the network. A bandwidth attack can also shift traffic to a less-secure part of a network, enabling other security attacks.

If an attacker simply sends as many packets as it can toward a victim, the resources expended by the attacker may be similar to the resources expended by the victim! A *distributed denial-of-service attack* can be launched from many coordinated machines, focusing the resources of many machines onto a smaller number of targets. Alternatively, a flooding attack can employ some form of *amplification,* in which the attacker's resources are amplified to cause the victim to expend far more resources. Here are some well-known forms of amplification:

- A "botnet" is formed by penetrating large numbers (as in millions) of innocent-but-buggy machines connected to the Internet, and installing in them a particular kind of malware. Subsequently, the attacker sends a triggering packet to each member of the botnet, causing it to launch a security attack unbeknownst to the machine's owner. This is another kind of distributed denial-of-service attack.
- An "asymmetric attack" sends requests to a server that require it to expend significant compute or storage resources for each request, so that a relatively small amount of traffic is sufficient to launch a significant attack. A typical IP example is a "SYN flood," in which the victim receives a flood of TCP SYN (session initiation) packets. Each packet causes the server to do significant work and allocate significant resources such as buffer space. Also, in IP networks, attackers can flood DNS servers with queries for random domain names (a "random subdomain attack"). These will force the servers to make many more queries, because they will have no cached results to match them.
- An attacker can send many request packets to public servers, with the intended victim's name as source name. This "reflection attack" causes all the servers to send their responses

to the victim. It amplifies work because responses (received by the victim) are typically much longer than requests (sent by the attacker).

Network infrastructure provides the principal defense against flooding attacks, by filtering out attack packets (Section 5). Flooding attacks can also be countered by allocating additional resources to handle peak loads (Section 6); this is something that both network infrastructure and targeted users can do.

If network infrastructure discovers where attack traffic is coming from, defending against the attack becomes much easier. For this reason, attackers employ various techniques to hide themselves, for example:

- In an IP network, a sender can simply put a false source name in the packet header, commonly called "spoofing." In email applications, source email addresses are also easily spoofed.
- With a botnet, even if bots use their true source names, there may be too many of them to cut off. The IP address of the master of the botnet remains hidden.
- An attacker can hide by putting a smaller-than-usual number in IP packets' time-to-live fields, so that the packets are dropped after they have done their damage in congesting the network, but before they reach a place where measurements are collected or defenses are deployed.

Flooding attacks are a very serious problem in today's Internet. There are businesses that generate them for small fees. They target popular Websites and (especially) DNS [9]. The worst attacks are mounted by enterprises, albeit illegal ones, that can draw on the same kind of professional knowledge, human resources, and computer resources that legitimate businesses and governments have. Such attackers will use many attacks and combinations of attacks at once, and can continue them over a long period of time. According to industry reports, we are entering the era of flooding attacks of terabits per second [1].

2.1.2 Subversion Attacks. The purpose of a subversion attack on a network member is to get the victim's machine to act as the attacker wants it to, rather than as the owner of the machine wants. Here are some well-known examples:

- The attacker sends malware to infect or penetrate the machine. The malware might be spyware or ransomware, capable of stealing or damaging data stored in the machine. The malware might turn the machine into a bot, so the botnet master can exploit the machine's resources. Or it might attack the physical world through devices controlled by the machine.
- Port scanning is the process of trying TCP and UDP destination ports on a range of IP addresses, to find pairs that will accept a session initiation. Port scanning does not in itself do much harm, but it is gathering information to be used in launching other malware attacks. This is because most malware targets a known vulnerability in a specific application program. Scanning is less productive in IPv6, because the address space is much larger, but specially focused scans may still succeed.
- The Border Gateway Protocol (BGP) is a control protocol through which IP networks exchange routing information. In "BGP hijacking," an attacker uses BGP to insert false information, telling routers to send packets with certain destination names to the attacker rather than the true destination. The attacker may simply drop the redirected packets, denying service to the victim. The attacker can also respond to the packets as an impersonation of the intended destination, for the purpose of stealing commerce or secrets.

- Subversion attacks on directories also insert false information. Higher-level names will then be mapped to the wrong lower-level names, with the same consequences as route hijacking. The directory protocol DNS (World-Wide Web name to IP address) and the IPv4 directory protocol ARP (IP address to Ethernet address) are subject to subversion attacks, as is the IPv6 replacement for ARP, called Neighbor Discovery.
- Email spam and voice-over-IP robocalls can be considered subversion attacks. A networked device's owner wants the device for communicating with acquaintances and chosen institutions. These attacks force the device to present ads and other unsolicited junk to the attention of its owner.

If a receiver of information knows the correct source of that information, then both users and network components can protect themselves from subversion by using cryptographic protocols. With cryptographic authentication, they know the identity of the agent with which they are communicating.

In other cases, network infrastructure protects itself and its users from subversion attacks by traffic filtering. But filtering for subversion attacks is significantly different from filtering for flooding attacks because subversion requires two-way communication between attacker and victim. For example, if the victim is a server that communicates using TCP, the attacker cannot send data to it until the initial TCP handshake is completed. This means that an attacker cannot hide by spoofing: if an attacker puts a false source name in its first packet to the victim, it will never receive a reply to its SYN, and can never complete the handshake.

2.1.3 Policy Violations. Obviously, the default behavior of a network is to provide all communication services requested of it. On the other hand, the administrative authority of the network, or other authorities such as governments, employers, and parents, may have policies constraining network communication. Specific communications that violate these policies are security attacks, and the network defends against these attacks by tampering with the communications (up to and including blocking them) or by spying on them so that other enforcement actions become possible. These defenses are exceptions to the default behavior of the network. Examples of policy violations include:

- Two users can communicate for the purpose of committing a crime. This should be prevented, or in some cases recorded for evidence in legal proceedings ("lawful intercept"). Similarly, the communications of suspected individuals can be monitored for surveillance and investigation.
- Saboteurs can attempt to access the control system of a power grid.
- A minor can attempt to access a Website that violates parental controls.
- A network may consider certain voice or video applications to take up more bandwidth than individual users are entitled to and rate-limit them to minimize their effects on overall performance.
- Operators of enterprise networks know which employees are using which machines for which purposes. Often they configure their networks to prevent unnecessary communications, which may be attacks, and can be blocked without harm even if they are only mistakes. For example, machines used by engineers should not have access to the enterprise's personnel database.

Network infrastructure defends against policy violations by traffic filtering. As indicated above, violating packets can simply be discarded, but they can also be recorded, tampered with, or rate-limited.

Traffic filtering for policy enforcement is different from traffic filtering against flooding and subversion attacks because the filtering is so specific. There is often a specified target whose communications are being monitored. Flooding and subversion attacks, in contrast, usually have unknown sources, and their victims are often opportunistic.

2.1.4 Spying and Tampering. The victims of spying and tampering are network users, who want their communications to be private, and want the network to be a transparent and effective medium of communication. The attackers in spying and tampering can be unsafe users, or they can also be the infrastructure machines of the network itself. Note that tampering is different from subversion because, in subversion, one endpoint of the communication is the attacker. In a tampering attack, the communication has two innocent endpoints, and the attacker is causing what one endpoint receives to differ from what the other endpoint sent.

When the attacker is the network, a spying or tampering attack is the exact dual of a policy violation—both the users and the network are doing exactly the same thing, and the only difference is which party we consider good or bad. Judgments of which behaviors are good or bad emerge from social debates involving legal, commercial, political, and ethical considerations. These debates should not be constrained by technology. Rather, the goal of technical experts should be to have the knowledge to implement whatever decisions emerge from these debates [7].

Examples of spying and tampering include:

- Some governments censor the Internet usage of their citizens. Even if networks in their countries are privately owned, the governments can insist that network providers enforce their policies.
- Some governments use surveillance of network usage for repression of or retaliation against political dissidents.
- By observing the searches and Web accesses of a network user, an attacker can learn a great deal about the user's personal life.
- Networks can insert into the paths of user sessions middleboxes that insert ads or alter search results.

Network users have two possible defenses against spying and tampering. The first is the use of cryptographic protocols (Section 4), which conceal the data in transmitted packets. The second is the use of compound sessions and overlays (Section 7), which seek to hide packets so that even their headers, sizes, and timing cannot be observed.

2.2 Relation to Other Definitions of Security

2.2.1 The Information-Security Triad. Governments, enterprises, and other institutions have broad concerns about information security. These concerns are articulated by the well-known "information-security (CIA) triad," consisting of the properties of *confidentiality* (secrecy, privacy, access control), *integrity* (the information is valid or uncorrupted or has correct provenance information), and *availability* (information can be read or written whenever needed).

These broad concerns about privacy include insider attacks and theft of physical storage media. The broad concerns about integrity and availability include natural disasters and even military attacks that might affect data centers. If the opposite of availability is denial-of-service, we can see that *denial-of-service attack* is also an extremely broad category.

Although the goals of the CIA triad have a great deal of overlap with the goals of network security, the classification scheme of Section 2.1 is far more focused. It is confined to threats incurred by operating a network or being connected to one, and it is closely tied to specific defense mechanisms within networks.

2.2.2 Complementary Forms of Security. For network users, network security is a first line of defense against subversion attacks; a major goal is to keep subversion packets from being delivered to user machines. If the packets do arrive, then security measures in operating systems and applications must take over. Many applications and most operating systems now have well-developed security measures of their own. However, old operating systems, real-time operating systems, and Internet of Things devices (which are highly resource-constrained) tend to have far fewer security mechanisms built in. For these endpoints, network defenses against subversion remain important.

Another subfield of security research and practice concerns "trust management," which is technology aimed at deciding which agents should have permission to access which resources or perform which operations, based on the credentials and attributes of the agent, and on the permission policies applicable to the object (see, for example, [13] and [23]). Trust management is a decision-making component of most forms of security, including network security.

Most security experts would probably agree that the human side of security is the most important and the hardest to deal with. In an ideal world, all institutions would have sophisticated cyber-security policies, and enforce them. These policies would prevent (among other problems) insider attacks in which employees with access to code deliberately put bugs or backdoors in it. All people using computers would keep their software updated, choose hard-to-guess passwords, and change default passwords immediately. (Botnets are heavily populated with Internet of Things devices such as baby monitors, because they come with factory-installed passwords, and their naive owners do not change them.) No one would be fooled by "phishing" attacks, which imitate a legitimate email so that the recipient clicks on a malicious hyperlink embedded in it. And on and on.

2.2.3 Threats with Inadequate Defenses. Personal data privacy is a form of security that is much discussed in today's world. Individuals are concerned about the massive amounts of personal data that is collected about them by Web sites, search engines, and other applications. This data is extremely valuable for selling advertising, and can also be used for worse purposes. Individual users can protect their privacy to some extent by using anti-spying defenses to achieve anonymity. Anonymously, they can email and participate in social media. At some point, however, full participation in electronic commerce and institutional services almost forces people to disclose their identities [37].

Finally, there is the growing threat of side-channel attacks. Network infrastructure monitors traffic to filter out flooding attacks, subversion attacks, and policy violations. Attackers also observe and analyze network traffic, for the purpose of spying and tampering. What are the characteristics of network traffic to be observed and analyzed, in addition to principal header fields and packet contents (which are explicitly intended and known to carry information)?

The timing and sizes of packets can be observed. Pseudo-random header fields, intended merely to group or distinguish packets, might be carrying secret codes. Optional header fields might reveal the configuration of the machine or software version that produced it. If the observer has access to the machine that sent the packet, it might be able to observe processor timing, power consumption, or usage of shared resources as the packet is prepared. Such access is possible if the machine is a stolen mobile device, or if multiple tenants share a physical machine in a cloud.

All of these characteristics are usually incidental, but they can be controlled by the sender to signal information to a knowledgeable observer that is invisible to other observers. This is known as a "covert channel." Incidental characteristics can also be analyzed by an adversarial observer, to gain information despite the intentions of the sender. This is known as "side-channel" information [36]. Extracting side-channel information from packet timing and sizes is becoming more common, both for (good) filtering and (bad) spying, because the expanding use of cryptography has hidden much explicit information [28]. At present, defenses against side-channel spying are patchy and experimental.

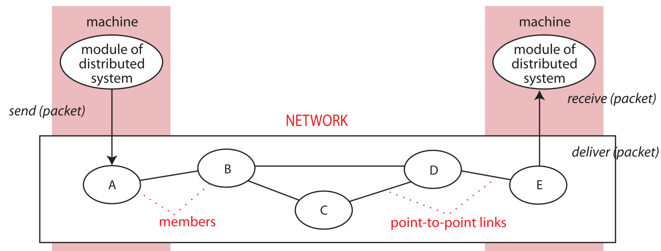

Fig. 2. Components of a single network, and its user interface.

3 A MODEL OF NETWORKING

To find the patterns underlying network security mechanisms, and to understand how these patterns interact with each other and with other aspects of network architecture, we must be able to describe today's networks in a way that is somewhat abstract and yet very precise. The "classic" Internet architecture [6] and the OSI reference model [17] have not kept up with the Internet's evolution since the early 1990s. For a better way to describe networks, we will use the compositional model of networking introduced in [40]. In this section we give a brief overview of the compositional model, covering the structures and aspects that will be used in the rest of the tutorial. Although the model uses familiar terms, be aware that when they have definitions within the model, it is these precise and specific definitions that apply.

3.1 Components of a Network

The components of a network are *members* and *links*. A *member* of a network is a software and/or hardware module running on a computing *machine,* and participating in the network. As a participant, the member implements some subset of the network's protocols. A network member usually has a unique *name* in the namespace of the network. For example, Figure 2 shows five members of a network with unique names *A, B,* and so on.

In the compositional model, a network always has a single *administrative authority*, or alternatively *network operator*, which is a person or organization responsible for the network. The operator provides and administers resources for the network, in the form of links, members, and additional resources on the members' machines. The operator is expected to protect the network's resources and ensure that users of the network enjoy the promised communication services. It is convenient to partition the members of a network into *infrastructure members* administered by the operator to provide services, and *user members* belonging to the network for the purpose of employing its services.

A network member can send or receive digital units called *packets* on one or more *links* of the network. A *link* is a communication channel. In this version of the tutorial, we only consider point-to-point links, as broadcast links are not fundamentally different for security.

A *public* network allows any machine to host a network member and connect to the network, while a *private* network allows only authorized members. The two common authorization mechanisms are cryptographic protocols (Section 4) and physical security, in which intruders are denied physical access to the links of the network.

3.2 Functions of a Network

As shown in Figure 2, a network enables modules of a distributed system on different machines to communicate. We say that a network provides one or more *communication services.* A particular instance or usage of a communication service is called a *session.* Like a link, a session is also a

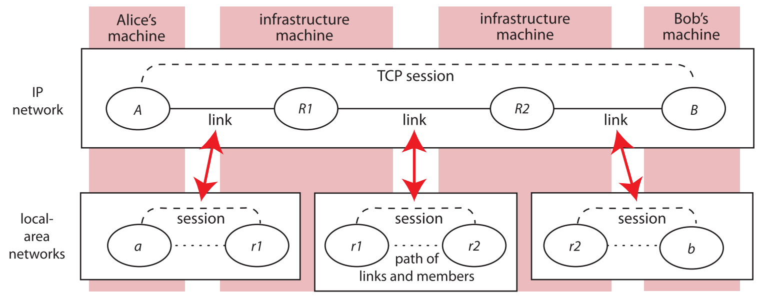

Fig. 3. The IP networks of the Internet are layered on many local-area networks.

communication channel for digital packets. The minimum semantics of a session is that it is a group of packets that the users of the service regard as belonging together.

There are two major mechanisms for providing network services. The first is *routing* and *forwarding*. *Forwarding* is the mechanism that extends the reach of the network beyond individual links to paths of links; in forwarding, a member receives a packet on an incoming link, and sends it out on an outgoing link to get it closer to its destination. A *forwarder* is an infrastructure member whose primary purpose is forwarding. Figure 3 shows a path through an IP network between user members *A* on Alice's machine and *B* on Bob's machine. In the figure, *R1* and *R2* are "IP routers," i.e., forwarders. All these names are "IP addresses."

Routing is the control mechanism that controls forwarding by populating *forwarding tables* in forwarders. Forwarders consult their tables to know where to forward packets. Routing and forwarding can be extended beyond minimum requirements of reachability to perform services such as broadcasting and steering packets through *middleboxes*—members that perform various packet-processing functions related to security, efficiency, or interoperability.

The other major mechanism for satisfying service specifications is *session protocols*. A *session protocol* is a set of rules governing packet formats, higher-level semantic units, and participant behavior during a session. In Figure 3, the session shown in the IP network uses the TCP session protocol. Following the rules of TCP, the session endpoints maintain state and send extra packets to provide reliable, ordered data delivery despite the facts that IP links are not perfectly reliable, and different packets of a session may be routed on different paths. UDP (another IP session protocol) is much simpler and implements fewer services, but it does define port numbers that can be used to group-related packets.

3.3 Composition of Networks

We have defined networks as self-contained modules with members, links, routing, forwarding, and session protocols. In today's Internet, there are many networks, each of which may be specialized according to its purpose, membership scope, geographical span, and level of abstraction. A network architecture is a flexible composition of these networks, and thus called a "compositional network architecture" [40].

There are two composition operators on networks, the first being *layering*. The model defines layering precisely: one network is *layered on* another network if a link in the overlay network is implemented by a session in the underlay network. For example, each IP link in Figure 3 is implemented by a session in a local-area network (see bold arrows). Members of different networks on the same machine communicate through the operating system and/or hardware of the machine. IP packets sent on an IP link are actually encapsulated in Ethernet headers and transported through local-area networks as the data parts of Ethernet packets. Since the implementation of an overlay link always consists of digital logic, whether in hardware or software, an overlay link is always virtual, regardless of whether the links in the underlay are physical or virtual. The IP network in Figure 3 plays the same role as the distributed system in Figure 2.

As Figure 3 shows, almost all networked machines host members of at least two networks, and some host many more. We use the term *member* rather than *node* because the latter is too similar in connotation to *machine*. The figure shows how layering extends the reach of the local-area networks, each of which is isolated. A local-area network only implements an IP link, but the IP network can reach machines over paths that are concatenations of links.

The second composition operator on networks is *bridging*. *Bridging* simply means that two particular networks share some links, so they can forward packets to each other. If the designs of bridged networks are sufficiently homogeneous, in particular if they share session protocols, then sessions can cross network boundaries. In the Internet, many IP networks are bridged together in this way. These networks differ in their operators/administrative authorities, but not their basic design.

The definition of layering in compositional network architecture is very different from the older notion of layering in networks found in the "classic" Internet architecture [6] and OSI reference model [17]. In the new model, each layer is a complete network, so IP routing/forwarding and IP session protocols belong to the same network/layer. In the new model, an architecture has as many layers as needed, which often includes multiple IP networks layered on top of one another. We use the compositional model in this tutorial because it allows comprehensive yet precise descriptions of how the Internet actually works today [40]. It is also necessary for recognition of the four patterns, because the same patterns are reused in different networks in a compositional architecture.

4 CRYPTOGRAPHIC PROTOCOLS

Cryptographic protocols are incorporated into the session protocols of a network. Cryptographic protocols are executed by the endpoints of a point-to-point session, so that the session will have *(data) integrity* and *(data) confidentiality*. These are the same terms used in the information-security triad, but in this context they have a much more specific meaning. Confidentiality means that no party except a designated receiver can read the packets sent. Integrity means that no third party can insert, modify, or replay packets of the session, so that the packets received by a designated receiver are the exact packets sent by the designated sender, and if the sender sends a distinguished packet m times, the receiver receives it at most m times.

Cryptographic protocols can also achieve *endpoint authentication*, which means that either session endpoint can be sure of the other endpoint's identity. Confidentiality should be reinforced by the property of *forward secrecy*, which means that even if an encrypted session is recorded by an attacker, and the attacker learns the secrets of one of its endpoints at some later time, the attacker still cannot decrypt and read the recorded packets.

User members of a network use cryptographic protocols to protect themselves against spying and tampering attacks. Infrastructure members, also, defend network operations against spying and tampering with cryptographic protocols.

It is important that cryptographic protocols are designed for the most hostile environments. For example, in accepted proof systems (such as [5] and [26]), the baseline model of a security protocol allows an adversary to control all communication channels between the endpoints (and other agents they might query), examining, storing, deleting, injecting, or altering any packets that the adversary wishes. Because cryptographic protocols are designed (and proved mathematically) with such conservative assumptions, users trust them even when they can trust nothing about the layers of networking between endpoints.

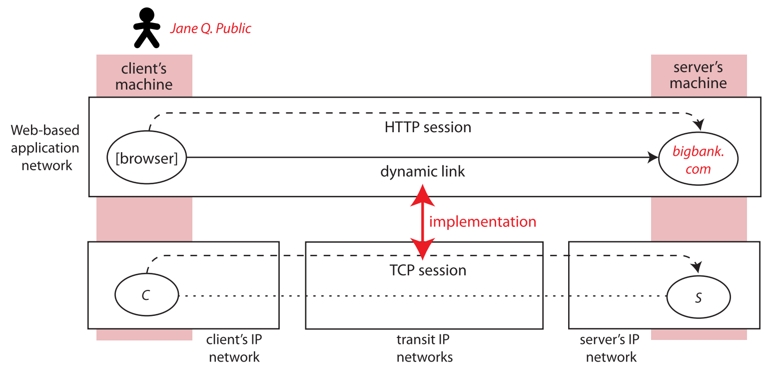

Fig. 4. Member names and identities in a Web application.

Section 4.1 begins our discussion of cryptographic protocols by introducing the central concept of *identity*. The foundation for all cryptographic protocols is public-key cryptography (Section 4.2), because it provides some crucial functions and supports others. In Section 4.3, we return to the properties of data integrity and confidentiality. Finally, in Section 4.4, we discuss architectural interactions with cryptographic protocols that are relatively independent of other security patterns.

In Section 4.1 through Section 4.3, the context will be a single network of any kind. The discussion also covers a set of similar bridged networks all at the same level of the layering hierarchy, for example the bridged IP networks of the Internet. Section 4.4 broadens the context, as it includes how cryptographic protocols interact with composition of networks by layering.

4.1 Trust and Identity

Security requirements are based on which network members do and do not "trust" each other. Of course, a network member is a software or hardware module; it cannot trust in any ordinary sense of the word, and has no legal responsibility that it can be trusted to fulfill. For the purpose of establishing trust, a network member that is an endpoint of a session has an *identity*. This identity is the answer to the question, "With whom am I communicating?"

This role implies that an identity should have meaning in the world outside the network. Often it is closely associated with a legal person—a person or organization—who is legally responsible for the network member. The identity is usually the source of the data that the network member sends during the session.

Identities are related to layering, because layering allows a machine to have different names—one in the namespace of each network it participates in—at the same time. For example, in Figure 4, each machine is participating in a higher-level Web-based application network and a lower-level IP network bridged with other IP networks. The dynamic sessions and links in the figure are formed as follows: The client's browser at the upper level instructs its IP member C to contact *bigbank.com*. When there is layering of networks, a *directory* is often used to find where an overlay member is *attached to* an underlay network. C looks up *bigbank.com* in the DNS directory, and finds it is located on the same machine as IP member S. At the lower level, C initiates a TCP session to S. When the TCP session (and dynamic link) are ready, the browser initiates a request/response HTTP session over it.

If the two endpoints of the TCP session need to authenticate each other (as they should, for a banking transaction), what identities do they give as their own? The general answer is that each gives its member name or the name of a higher-level network member that is using it. Either IP interface could give its IP name, but it would not be a very good identifier—too transient, or

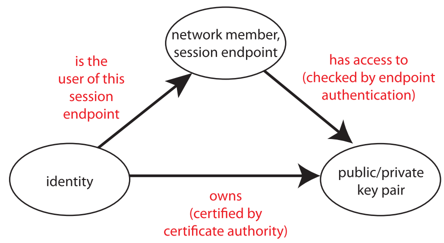

Fig. 5. Relationships among identification entities.

with too little meaning in the outside world. Instead, the server's IP interface S will be known by its public Web name *bigbank.com*. The client's machine does not have a name in the application network, because the browser only initiates sessions and never accepts them. However, the user of the browser is a person named *Jane Q. Public*, whose clicks and keystrokes provide input to the browser. The browser will send *Jane Q. Public* as its identity, and we can imagine this identity as a member of an even-higher-level distributed financial system.

For endpoint authentication, a member must have access to a secret associated with the identity it provides. One kind of secret, useful when the two endpoints have an ongoing relationship, is a password. The server *bigbank.com* knows Jane's password, and she can type it into the browser when requested.

For the important cryptographic protocols, however, the secret is always a public/private cryptographic key pair (see next section). The relationships among the important entities are shown in Figure 5. The identity is responsible for the packets sent by the network member, and the network member has access to the public key and its paired private key.

A "certificate authority" is trusted to ascertain that a particular public key belongs to a particular identity; it issues a certificate to that effect and signs it digitally. Thus, when an endpoint receives a certificate, it can trust the identity that goes with the key (at least, as well as it trusts the certificate authority). As indicated above, identities found in certificates include names of legal persons, domain names, and IP addresses.

4.2 Public-Key Cryptography and Its Uses

In public-key cryptography, an identity generates and owns a coordinated pair of keys, one public and one kept private and secret. The important properties of these keys are that (i) it is extremely difficult to compute the private key from the public key and (ii) plaintext encrypted with the public key can be decrypted with the private key, and vice-versa. Today's public-key cryptography is descended from the Diffie-Hellman Key Exchange protocol and the RSA algorithm (named for its inventors Ron Rivest, Adi Shamir, and Leonard Adleman). At present, a key must be at least 2,048 bits to be considered secure, and the minimum size is expected to increase in the future.

4.2.1 Endpoint Authentication. A simple challenge protocol is sufficient to determine that an endpoint has access to a public/private key pair. Suppose that an endpoint B is engaged in a session with endpoint A, and wants to check its identity's claim to own public key K^+. B can make sure of this by sending a *nonce* (a random number used only once in its context) n. A is supposed to reply with $K^-(n)$, which is n encrypted using the private key K^- that goes with public key K^+. B then decrypts the reply with K^+. If the result is n, then B has authenticated that the other endpoint indeed has access to public key K^+ and its private key K^-.

In practice, B may not know the public key ahead of time. In a typical client/server protocol, the client needs to authenticate the server, but the server does not authenticate the client. The

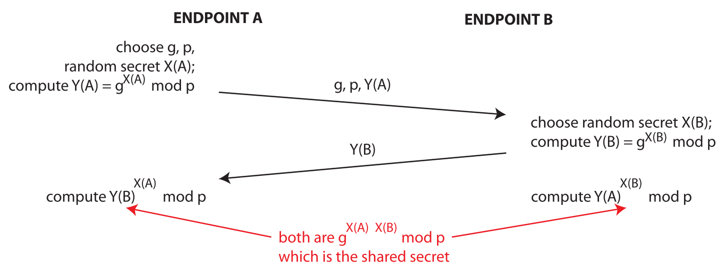

Fig. 6. Diffie-Hellman key exchange. g is a small number such as 2 or 3, while p, $X(A)$, and $X(B)$ are large integers.

client B might send its nonce to A, and A might reply with both its certificate and $K^-(n)$. From the certificate, B gets K^+. The client should validate the certificate as well as the encrypted nonce, including checking that the identity in the certificate is the identity expected, checking that the certificate has not expired, and checking that it has been signed by a legitimate certificate authority. Some client software validates certificates poorly or not at all, causing it to be dubbed "the most dangerous code in the world" [11].

A server can delegate its identity to another trusted network member, by giving the delegate its certificate and keys. For example, "content-delivery networks" host Web content on behalf of other enterprises. Content-delivery servers are trusted delegates of their customers, and each such server can have many delegated identities.

4.2.2 Digital Signatures. A digital signature transmitted with a document can be checked to verify that the document came from a specific identity, and has not been modified in transit. The simplest digital signature of a document m would be $K^-(m)$, i.e., the document itself encrypted with the private key of the signer. The recipient decrypts the signature with the public key of the signer. If the result is m, then the signature and document are verified.

Because public-key encryption is computationally expensive, encrypting whole documents would be very inefficient. Instead a *cryptographic hash* is used. The hash function H is computed from a digital message m (of any length) to a fixed-length bit string. Its important property is that, given a hash $H(m)$, it is extremely difficult to compute a different message m' such that $H(m) = H(m')$. So a (short) cryptographic hash $H(m)$ of the document can be encrypted with the private key and used as a digital signature. To verify the signature, the recipient both encrypts the signature with the public key, and computes the same hash function on the plaintext document. Verification is successful if they match.

If a client is interested in the identity of a server only to obtain its authentic data, then receiving data signed by the server is just as good as receiving data directly from the server.

4.2.3 Key Exchange. Because public-key cryptography is computationally expensive, it is used only to encrypt small amounts of data. For encrypting the entire data stream being transmitted on a link, *symmetric-key cryptography*, which is much more efficient, is used. As the name implies, symmetric-key cryptography requires that both endpoints have the same secret key, which is used to both encrypt and decrypt the data. This raises the problem of "key exchange," or how to distribute secret keys securely over insecure channels. The basic solution to the problem is the Diffie-Hellman algorithm, shown in Figure 6.

Unfortunately, the basic algorithm is vulnerable to a "man-in-the-middle" attack, which refers to any attack carried out by an adversary able to intercept packets on a link. The adversary can read, absorb, inject, or alter any packet transmitted on the link; the attacker can also "replay" packets by storing them and retransmitting them later. Figure 7 shows how such an attack would work.

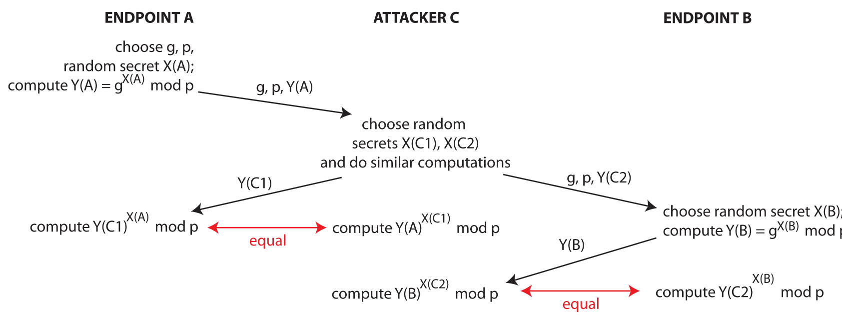

Fig. 7. A man-in-the-middle attack on Diffie-Hellman key exchange.

The adversary simply engages in a separate key exchange with each of the two endpoints. After the key exchange, the adversary can relay packets transparently between A and B by decrypting with one key and encrypting with the other; it can also read the packets and manipulate them in any way whatsoever.

Fortunately, the solution to this problem is straightforward. A and B must have identities and public/private key pairs, and must authenticate each other before the key exchange. Then protocol packets must bear the sender's digital signature. Even if the attacker can read $Y(A)$ and $Y(B)$, it can do nothing with them.

4.3 Three IP Cryptographic Protocols

This section provides an overview of security in the three most important cryptographic protocols in the IP suite:

- Transport Layer Security (TLS) is the successor to Secure Sockets Layer, and is an extension of TCP. Two versions of TLS, 1.2 and 1.3, are in widespread use.
- Quic [22] is a new protocol proposed as an alternative to TLS. Its security mechanisms are similar to TLS 1.3.
- "IPsec" refers to a family of related IP protocols, comprising the Authentication Header and Encapsulating Security Payload (ESP) protocols, each of which can be used in "transport mode" or "tunnel mode." ESP is more useful than Authentication Header, so only ESP will be discussed here.

These protocols provide endpoint authentication, data integrity, data confidentiality, and forward secrecy. They have interesting differences, and the differences are significant for their use in compositional network architectures.

4.3.1 Protocol Embeddings. Within a network, session protocols can be composed, so that the same session benefits from the services implemented by multiple protocols. When two session protocols P and Q are composed, one of them is *embedded in* the other.

TLS is composed with (embedded in) TCP. If the Uniform Resource Locator (URL) of a Web site begins with `https://`, then its clients should make requests of it using IP protocol TCP and destination port 443, signifying the use of TLS embedded in TCP. Figure 8 shows packet formats for TLS, ESP in transport mode, and ESP in tunnel mode. Fields are labeled to show which headers belong to embedded and embedding protocols.

When ESP is used in composition with TCP in transport mode, TCP is simply embedded in ESP. In contrast, ESP in composition with TCP in tunnel mode is an instance of layering (recall Section 3.3). An entire overlay packet with IP/TCP headers and data is encapsulated in the data part of an underlay IP/ESP packet. So the important distinction between ESP transport mode

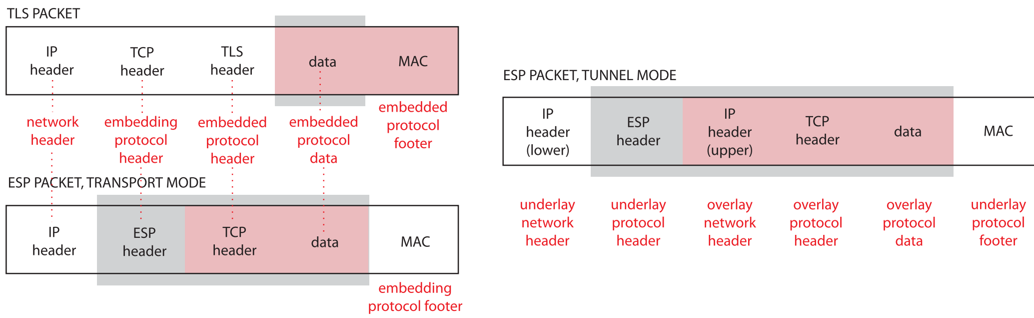

Fig. 8. Packet formats for cryptographic protocols, slightly simplified. The Message Authentication Code (MAC) is a footer that assists in message authentication. Pink parts of a packet are encrypted, while gray parts are authenticated.

(session-protocol composition with TCP) and ESP tunnel mode (layering composition with TCP) is that, in tunnel mode, there is an upper IP header with a completely different destination than in the lower IP header. Intuitively, the upper destination is the ultimate destination of the TCP session, while the lower destination is the next hop in the session path (see Section 4.4.1).

Quic is embedded in UDP, and also uses destination port 443. When a client accesses an `https://` Website for the first time, it should use TLS. If responses carry an "I support Quic" code, subsequent requests from that client to that server should use Quic, with TLS as a fallback in case of problems.

4.3.2 The Setup Phase. In a TLS 1.2 session, the client and server first have a TCP (control) handshake, in which they establish the session identifier and other parameters. They then begin a TLS 1.2 (control) handshake, which performs three tasks: (i) endpoint authentication (Section 4.2.1), (ii) negotiation of a "cipher suite," and (iii) key exchange Section 4.2.3). Usually, the accepting endpoint is authenticated with a certificate and the initiating endpoint is not, because the acceptor is a server and the initiator is a client.

TLS supports many different methods for exchanging keys, encrypting data, and authenticating message integrity (see below). For each of these tasks, there are many possible algorithms (counting all variations of a few basic algorithms). A "cipher suite" is a collection of algorithms and parameter choices for doing all the cryptographic tasks within a security protocol. The most important parameter choices govern key length, because key length has a big effect on the overall security of cryptography. To negotiate a cipher suite, the initiator sends all the cipher suites it implements, and the acceptor chooses one that it also implements and sends back the choice.

The TLS 1.2 handshake adds two round-trip times for TLS setup on top of the one round-trip for TCP setup. Slightly simplified, there is one round-trip for authentication and negotiation, and one for key exchange. The property of *forward secrecy* is achieved because fresh symmetric keys are computed for each session.

Security in TLS 1.3 is very similar to the security in Quic. One difference between TLS 1.2 and Quic (or TLS 1.3) is that Quic disallows some older cipher suites that are known to be insecure, and requires longer keys. Another difference is that Quic/TLS 1.3 setups are faster than TLS 1.2 setups. For faster setups, Quic combines the initial transport handshake with the initial security handshake. After this there is one additional round-trip for key exchange. Further, the key-exchange round-trip can be combined with the first data round-trip, because the client's first data request is allowed to use a less-secure symmetric key; the server's first response and all subsequent data packets are encrypted with the final, secure symmetric keys. Even further, this one-round-trip setup can be eliminated entirely if the client has saved authentication and negotiation information from previous contact with the server. In this "zero round-trip" setup, the first round trip combines data and key exchange as above.

ESP endpoints authenticate each other if required, negotiate cipher suites, and exchange keys by means of the Internet Key Exchange (IKE) protocol. The result is that each ESP endpoint has long records called "security associations" including choices of cipher suite and actual keys. Use of full IKE to set up an ESP session is not always necessary because security associations can also be introduced into ESP endpoints by configuration, or saved from previous negotiations. Needless to say, if perfect forward secrecy is required, longer-term parts of a security association can be re-used, but there must be a new key exchange for symmetric keys.

4.3.3 Data Integrity and Confidentiality. In all three protocols, data and some headers are encrypted with a shared key by the sender, and decrypted using the same key by the receiver. A different shared key is used in each direction. According to the mathematics of symmetric-key cryptography, encryption satisfies the requirement of data confidentiality.

The requirement of data integrity is satisfied by the process of "message authentication." Each packet is sent with a "message authentication code" (MAC) computed from the authenticated data d by appending to the data a shared authentication key k, and then applying a cryptographic hash function (Section 4.2.2) to $d + k$. The MAC $H(d + k)$ is then appended to the data in the packet. As with encryption keys, all three protocols generate authentication keys during key exchange, and use a different authentication key in each direction. The packet receiver performs the same MAC computation and expects it to result in the same MAC that it received in the packet. If an attacker inserts or changes packets while they are being transmitted, it will not be able to compute correct authentication codes for the packets, and the discrepancy will be detected by the receiver.

This algorithm alone has the limitation that an attacker with access to the packet stream can still delete, re-order, or replay packets, even though it cannot create new ones. TLS and ESP require different solutions to this problem, because of the differences in embedding visible in Figure 8.

One might think that this problem would be solved for TLS (both versions) by the fact that the enclosing TCP packets have byte sequence numbers. TCP headers are not encrypted, however, so an attacker-in-the-middle could alter them to make even an altered TCP byte stream look correct. The actual TLS solution is for each endpoint to keep track of packet sequence numbers as TLS packets are sent and received. The sequence number is not transmitted directly, but it is included in the bit string hashed to compute the MAC. For a packet to be accepted, the receiver must be re-computing its MAC with the same sequence number that the sender used. This works because TLS is embedded in TCP, so the authenticated data and MAC are presented to the authenticator reliably and in sending order.

Message authentication in ESP and Quic must work differently, because their packets may not be presented to the authenticator in sending order. In these protocols, the headers contain explicit packet sequence numbers, which are included in the data on which the MAC is computed. The authenticator cannot predict the sequence number of the next packet it will see, so it cannot detect deletion or re-ordering attacks (which, after all, might not be attacks but flaws in the network). Rather, authentication checks only for received packets with sequence numbers that have already been received, and deletes them. This is sufficient to defend against replay attacks, which are part of many man-in-the-middle attacks, because an attacker cannot change the sequence number of a packet it replays.

4.3.4 Usage of Cryptographic Protocols. Almost all Web traffic is now encrypted, at least with TLS 1.2. Deployments of TLS 1.3 and Quic are both growing rapidly, because of the motivation of shorter setup times. TLS is also widely used by other application protocols. ESP is most commonly used to make "virtual private networks" (see Section 7).

Although cryptographic algorithms and protocols are proved mathematically, there is a big difference between mathematical abstractions and code. In implementing the algorithms, efficiency is a top priority, and transformations for efficiency can introduce bugs in addition to all the other bugs to which software systems are prone. Advances in processor speeds and the exploitation of side-channels are making it easier to crack codes, so that increases in key lengths become necessary—not even counting the unpredictable disruption that might be caused by quantum computing. Cryptographic libraries are improved continually, but each machine is no more secure than its latest upgrade. It may even be less secure, when it must use an older software version to communicate with an infrequently updated machine.

4.4 Interactions between Cryptographic Protocols and Other Aspects of Networking

Cryptographic protocols have significant interactions with other security patterns, which will be discussed when the other security patterns have been presented. This section is concerned with the interactions of cryptographic protocols with network architecture and network services other than security.

4.4.1 Layering. A network with cryptographic session protocols can be layered on top of one or more networks, as explained in Section 3.3. Because each underlay level can implement an overlay link with a path of links, forwarders, and middleboxes, users of an overlay network must accept that its packets can pass through many machines and physical links unknown to them. But cryptographic protocols are designed to work in completely adversarial environments such as these! Furthermore, the cryptographic properties of a session can be assumed to hold for any link that it implements, so the properties guaranteed by cryptographic protocols propagate upward through layering.

4.4.2 Performance. Data encryption and message authentication increase required bandwidth and computational resources. The overhead is modest, so it is not a concern in all cases. It is more likely to be a significant concern for battery-operated devices, or for network elements that must decrypt and re-encrypt at high traffic volumes.

The most direct and significant performance costs of cryptographic protocols are incurred in the setup phase, by endpoint authentication and key exchange, which consume compute resources and increase latency. Even with short round-trip times, a small fraction of TLS 1.2 setups take 300 ms or more [29], due to increased computation time. We have seen that newer protocols have reduced setup times aggressively, often by saving and re-using session state, but this causes an inevitable loss of security [33].

The performance issue is much more serious in applications for the Internet of Things (IoT), because these applications tend to have periodic or irregular short communications from a large number of networked devices to centralized analysis or publish/subscribe servers. Message Queuing Telemetry Transport, a protocol for IoT applications, is well designed from this perspective, because many short application communications can share the same TLS session.

For Message Queuing Telemetry Transport and all other application protocols with short or bursty communications separated by intervals of inactivity, it is most efficient for many communications to share a single, long-lived secure channel. Long-lived Internet channels have been difficult to maintain in the past, because various components in the path of the channel would time out and close the channel during intervals of inactivity. It is easier now—TCP, TLS, and DTLS all have keep-alive options, sending periodic keep-alive signals to keep long-lived channels open.

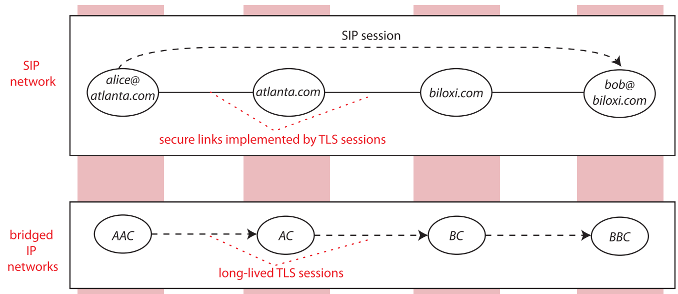

Fig. 9. A SIP application network layered on bridged IP networks.

Architecturally, there are two ways to implement the optimization of sharing a secure channel. The first way is to embed the application protocol in the security protocol. For example, if TLS is the security protocol, application headers and data would be the data portion of TLS packets, as shown in Figure 8. Alternatively, an application network could be layered on IP networks, as shown in Figure 9. The Session Initiation Protocol (SIP) is an application protocol for control of multimedia applications. The SIP application network has links that are implemented by TLS sessions in the IP underlay. The big difference between this architecture and protocol embedding is that the TLS sessions have different endpoints (different sources and destinations) than the SIP application session does. This makes it more flexible than protocol embedding, for two reasons: (i) The application network can insert its own middleboxes into the path of the application session, as SIP always does. (ii) The TLS sessions can last longer than any communication between two specific SIP endpoints, and can be shared by communications between many SIP endpoint pairs. Recall that embedding and layering correspond to ESP transport mode and tunnel mode, respectively.

4.4.3 Mobility. In its strongest sense, mobility enables a session to persist even though the network attachment of an endpoint device is changing. This usually means that, at some level of the layering hierarchy, the network member on the device is changing names within its network, or dying and being replaced by a member of another network. For example, when a mobile phone moves from one cellular provider's network to another, its IP name (for data service) must change. Ideally the data sessions of the phone would persist across such moves, as its voice sessions do.

There is usually no interaction between mobility and cryptographic protocols, because the identity of a mobile machine is at a higher level than the names that change. For instance, consider a Web server running on a virtual machine in a cloud. Because of failures or resource changes, the virtual machine may migrate to a different physical machine where it has a different IP name. But the identity of the Web server is its domain name, which is at a higher level and does not change. Similarly, more than one server can have the same identity, as when a Website of origin delegates its identity to a content-delivery server by sharing its certificate and keys.

On the other hand, thinking about mobility brings up the possibility of normal mobility in reverse—the higher-level identity moves or changes while the lower-level name remains the same. This can be a security issue: after *Jane Q. Public* enters her password (Section 4.1), she might walk away from her machine, and then any other person who walks by could retrieve her personal data and request transactions on her bank account. For this reason, secure distributed applications require periodic re-authentications of the identity of the person using them, especially after idle periods.

4.4.4 Infrastructure Control Protocols. Control protocols are used by network infrastructure to maintain and distribute network state. It is important to protect these protocols against subversion attacks (Section 2.1.2).

Unsurprisingly, some control protocols incorporate cryptography. For instance, Border Gateway Protocol Security is a security extension to BGP that provides cryptographic verification of packets advertising routes. Similarly, Domain Name System Security Extension protects DNS lookups by returning records with digital signatures.

In many cases, however, it is difficult for control protocols to rely on cryptography. An endpoint might not have a certificate or other credential to prove its identity. The protocol might require high-speed, high-volume operation. Or, the protocol might simply be too old to incorporate cryptography, even if it is feasible.

In these cases, there are lighter-weight measures that can help. Network members that make requests should keep track of their pending requests and not accept unsolicited replies. Replies should be checked for credibility, whenever that is possible. Most effectively, a network member can include a nonce or random field value in a session-initiation or request packet. Subsequent packets of the session must have the same nonce or random value, so that no attacker without access to the previous packets of the session can send packets purporting to be part of it. Without the nonce, an attacker could do something to trigger a query, then send a spurious answer to the query.

5 TRAFFIC FILTERING

Traffic filtering is performed by forwarders and middleboxes that are part of a network's infrastructure. The network's routing ensures that designated traffic passes through one of these *traffic filters,* and the filter examines it for evidence of flooding attacks, subversion attacks, or policy violations. If traffic seems to be part of an attack, the filter takes some defensive action, most often simply discarding the traffic.

Content-based traffic filtering (Section 5.1) looks at the contents of individual packets or sequences of packets. Path-based traffic filtering (Section 5.2) adds to this information about the paths along which traffic has traveled.

As in Section 4, the usual context of the discussion in this section will be a single network of any kind, or a set of similar bridged networks such as the bridged IP networks of the Internet. After explaining traffic filtering in individual networks, we return to the compositional view (Section 5.3), considering how traffic filtering interacts with other network mechanisms and where it should be placed in a compositional network architecture.

5.1 Content-based Traffic Filtering

5.1.1 Signature-based Filtering Criteria. Filtering criteria are predicates used to identify suspected traffic. Signature-based filtering criteria examine specific header and data fields of packets. These criteria are used to detect most policy violations and subversion attacks. Often the criteria are Boolean combinations of simple predicates such as *destinationPort = 80* on the values of IP header fields. For example, suppose that an IP "firewall" (traffic filter) at the edge of a network is enforcing this policy: the only external traffic is for Web accesses, which of course require DNS queries. The direction of a packet (inbound or outbound) can be determined from its source and destination names or from the link on which it arrives. The firewall might be configured with these four rules:

(1) Drop all outbound TCP packets unless they have destination port 80.
(2) Drop all inbound TCP packets unless they have source port 80 and the TCP ACK bit is set.
(3) Drop all outbound UDP packets unless they have destination port 53.
(4) Drop all inbound UDP packets unless they have source port 53.

	ROUTER	FIREWALL	INTRUSION DETECTION SYSTEM	INTRUSION PREVENTION SYSTEM
FILTERING CRITERIA	predicates on IP packet headers	predicates on IP packet headers; can have a table of ongoing sessions	any	any
REQUIRE SESSION AFFINITY?	no	yes if stateful	yes	yes
ACTIONS TAKEN	drop packets	drop packets	raise an alarm, divert packets for further analysis	drop packets, rate-limit packets, refuse requests, record packets

Fig. 10. Examples of common traffic filters in IP networks.

In the second rule, the ACK bit indicates that this packet is an acknowledgment of a previous packet, meaning that it is not a TCP SYN packet.

These rules are sufficient for the purpose if all packets through the firewall obey the TCP protocol exactly, but of course an attacker may not be so polite. A safer approach would be to make the firewall stateful by having it maintain a table of all ongoing TCP connections. Then the second rule above would be replaced by "Drop all inbound TCP packets unless their source and destination names and ports identify them as belong to an ongoing TCP session." If a firewall is stateful, it is crucial that all packets of a session pass through the same firewall. This property is called "session affinity."

For reference throughout Section 5.1, Figure 10 is a table summarizing characteristics of four common types of traffic filter in IP networks. The classification is at least as much historical and marketing-oriented as it is technical! It is a list of products that have sold well in the past, not a prescriptive list of which options are possible.

IP routers sometimes do dual duty as traffic filters. To do this, they are configured with predicates on packet headers, called "access control lists." Routers must work even faster than firewalls, so they do not perform stateful filtering.

For filtering that looks at packet data as well as headers, networks often use commercial products known as "intrusion detection systems" and "intrusion prevention systems." These filters can use any filtering criteria for any purpose. Signature-based filters against spam and viruses look for keywords, sometimes keywords in specific positions, and other known attack patterns. Their criteria can include regular expressions matching fields of arbitrary length. They can also be stateful, and check whether protocols are being followed. These filters can be valuable commercial products because of the intellectual property in their filtering criteria. Like all security software, to be effective, they must be kept up-to-date.

In the common case that TCP sessions are being filtered for subversion attacks or policy violations, the filter should reconstruct the correct byte stream (restoring packet order, replacing lost bytes by retransmitted ones) before filtering. If there is no reconstruction, attackers can hide attacks simply by splitting attack data over multiple packets. Even if there is reconstruction, there may be ambiguities exploitable by attackers. For example, if there are missing packets, some bytes may be retransmitted and received twice. An attacker can engineer the transmitted stream so that some bytes will have to be sent twice, and place attack bytes only in the second transmission. The filter might check only the first bytes, and the receiver might use only the second bytes. The surest way to avoid all such ambiguities is to have a "traffic normalizer" middlebox in the session path, before both filter and destination, to reconstruct a single unambiguous packet stream received by both of them [15].

One advantage enjoyed by TCP filters is that attacks require communication in both directions. Consequently, attackers cannot easily hide by giving false source names—if they did, there would be no two-way commnication. The sources of flooding attacks, on the other hand, can hide themselves behind false source names. This problem is also an opportunity, because having a false source name is a good indicator that a packet is part of a flooding attack. Forwarders (and other filters associated with them) are well situated for using this as a filtering criteria, because forwarders have information about routing. For example, "ingress filters" in IP networks check incoming packets to see if the prefixes of their source names match expectations. This is an excellent addition to an access network, which may have detailed knowledge of its user members, or an Internet service provider's network, which knows the IP prefixes allocated to each access network bridged with it. "Unicast reverse path forwarding" in a forwarder accepts a packet's source name as valid only if its forwarding table specifies forwarding *to* the source name on the same two-way link on which the packet arrived. Unfortunately, reverse-path checking cannot be used in the high-speed core of the Internet, because routes there are not necessarily symmetric.

5.1.2 Measurement and Statistical Analysis. Signature-based filtering criteria have two major limitations: they cannot detect new (called "zero day") attacks, and it is difficult to use them to detect flooding attacks, whose individual packets look normal (with the exception of false source names). In response to these limitations, forwarders collect data on large amounts of traffic, and send it to other network members for analysis. Analysis can measure attributes over large collections of packets. It can then look for known attack patterns, especially of flooding attacks; for example, a single destination receiving a large number of response packets from many different sources may be the victim of a reflection attack (Section 2.1.1). Analysis can also detect anomalies, which are new divergences from normal traffic patterns that may indicate new attacks. Anomaly detection uses statistical algorithms, including machine learning.

For typical traffic measurement in IP networks, routers collect selected data and send it to analyzers in some well-known record format such as NetFlow or IP Flow Information Export (IPFIX). Data can be collected at multiple locations and different levels of granularity. The volume of data can be reduced by recording only headers (rather than entire packets), by sampling the packets (rather than collecting information about all packets), or by focusing on specific subsets of the packets. Most importantly, a *flow* comprises a group of packets close together in time that have various header fields in common. Creating a single record for a flow helps reduce the volume of data while still providing a timely and detailed view of the network traffic.

Anomaly detection is a very attractive idea, but it is also very difficult in practice. One major reason is that normal Internet traffic is highly variable, not to mention unusual-but-innocent occurrences such as congestion due to failures, or a legitimate flash crowd [25]. The other major reason is that the cost of mistakes ("false positives") is high, as many legitimate packets are discarded. The best use of anomaly detection may be to discover and understand new attacks, then turn their characteristics into signatures or measurable patterns [35].

5.1.3 Defensive Actions. Obviously, the most common defensive action that a filter can take is to drop packets, but there are other possibilities.

The only difference between "intrusion detection systems" and "intrusion prevention systems" is that detection systems only raise alarms, while prevention systems automatically take action against suspected attacks. It might seem that automatic action is always better (it is certainly faster), but there are good reasons for keeping operators and enterprise customers in the decision loop. If a suspected attack is a false positive, much legitimate traffic may be dropped. If an operator deploys additional resources on behalf of an enterprise customer that is under attack, the customer will have to pay for them. In rare cases, the defense against a suspected attack may even be a counter-attack, which is wrong and even dangerous (in a military setting) if not well justified.

TYPE OF PACKET FILTER OR SERVER	APPROXIMATE CAPACITY
target server	1 - 10 Gbps / core
intrusion prevention system (reconstructs byte stream)	1 - 20 Gbps / core
stateful firewall (examines headers only)	20 Gbps / core
IP router with access-control list	100 Gbps / link

Fig. 11. Data-processing capacities of common traffic filters and servers.

What actions are normally taken by intrusion prevention systems, other than dropping packets? If there is uncertainty about the packets, a filter can rate-limit them or downgrade their forwarding priority rather than dropping them. Rather than dropping session-initiation requests, a filter could reply to them with refusals, which would discourage retries. A refusal to a TCP SYN (request) is a TCP RST (reset). A refusal to an HTTP request is an error code.

Finally, when filtering is being used to defend against policy violations, sometimes the filter records packets for the purposes of investigation and legal evidence. Recorded packets are usually *not* dropped but forwarded on to their destinations, to keep the investigation secret from its targets until it has been completed.

5.1.4 Resources and Capacity. Traffic filtering expends a lot of network resources, so the detailed design of a traffic-filtering mechanism must be resource-sensitive. How does a network ensure that its traffic filters do not themselves become traffic bottlenecks during flooding attacks?

There are two approaches to providing sufficient filtering capacity. The first is to host traffic filters on high-capacity machines dedicated to this purpose. The second is to run traffic filters on virtual machines in a cloud. In this approach, it is possible to implement "dynamic scale-out," which means that as the load increases during an attack, more virtual machines are allocated for the filtering task. All the filter types in Figure 10 have been implemented with both approaches, although the high-capacity-machine approach is more often applied to routers and firewalls.

The situation today is fluid, as flooding attacks are becoming more severe. A recent flooding attack on a number of DNS servers [9], including amplification, generated traffic at 10-20 times normal volume, with bursts up to 40-50 times normal volume, and reportedly a maximum of 1.2 Tbps (1,200 Gbps). To provide some intuition about the resources needed to handle such attacks, Figure 11 shows some typical capacities for servers and various kinds of traffic filters.

Of course, these numbers are subject to frequent change. On the positive side, converting an algorithm from software to programmable hardware increases its speed by a factor of about 10, as does converting it from programmable hardware to fixed-function hardware. On the negative side, commercial intrusion-prevention systems frequently fall short of advertised capacity. There are adversarial workloads designed so that the TCP byte stream is especially difficult to reconstruct. More commonly, rule-checking is the performance problem, because it is more expensive than reconstructing the byte stream; performance is improved when necessary by dropping rules.

5.2 Path-based Traffic Filtering

From the viewpoint of a victim of a flooding attack, the network is a tree with many possible packet sources and a single packet sink at the root. From the viewpoint of a packet source, its *access network* is the first network in all its outgoing paths whose administrative authority is different from the owner of the machine (assuming for simplicity that that there is only one such network). The access network of a machine is significant because it is the first network that is able to filter outgoing packets of the machine. Often the machine belongs to its access network, but the machine might belong to a home network, which carries its packets to its access network.

Path-based traffic filtering augments filtering criteria based on the contents of packets, as discussed in the previous section, with criteria based on the path along which packets traveled. There are two reasons for introducing path-based filtering. The first reason is that path information can improve the precision of filtering criteria, so that fewer good packets are accidentally included. For example, say that the overall load on a server suggests a flooding attack, and intrusion detection proposes a candidate filtering rule based on packet contents. If we know that most paths to the server are delivering a trickle of these packets, and one path's load is dominated by these packets, there is a good chance that only the packets on the dominated path are attack packets. The second reason is that—with knowledge of where packets came from—traffic filtering can be moved from its usual downstream location, near the target, to upstream locations close to packet sources. Upstream filtering has three main advantages:

- If filtering is farther from the target, the damage done by attack traffic is lessened, because attack traffic is carried for shorter distances along fewer links. Note that the damage of a flooding attack is not limited to the intended target, because traffic to many other destinations will also suffer because of congested links.
- If a traffic filter is close to sources of attack traffic, it may have more information about the sources. The access network sees all of a suspected source's traffic, so attack patterns are more likely to be detectable. An access network may also know more about the type and reputation of its sources (device type is relevant because some operating systems and vendor hardware are more easily penetrated than others). More precise filtering means less collateral damage.
- Very often, attack packets are coming from a botnet, with a large number of sources well-distributed across the public Internet. So the total amount of available filtering resources near sources greatly exceeds the total amount of resources available near a target.

The principal disadvantage of upstream filtering is that in IP networks, the source name in a packet is not a reliable indicator of where it came from. Forwarders can attach metadata to packets so that servers near targets can reconstruct packet paths, but this is problematic because of the volume of data involved and the danger of adversarial interference. All the proposals for "traceback" of this kind must make difficult trade-offs to balance the costs and benefits [2, 25, 32, 39]. In addition, upstream filtering has two other disadvantages:

- Upstream networks may not have sufficient incentive to use their resources to protect targets that are remote from them. It has been argued that networks under attack might be more willing to accept incoming packets from cooperating upstream networks, which will give the users of the upstream networks better service [2]. Historically, however, cooperation between the networks of different operators has been scarce [14].
- Even if source networks are willing to cooperate with target networks, the necessary coordination is not easy. Like traceback, coordination along packet paths invites its own security attacks.

Currently, the net effect of all these factors is that path-based traffic filtering is uncommon in the Internet. However, future changes might cause the factors to be weighted differently. For example, individual IP networks (under single ownership) now seem to be growing in size and geographical scope. If this trend continues, it will become more common for both the upstream and downstream segments of a path to an attack target to be controlled by the same operator. If so, then the administrative barriers to upstream filtering will disappear.

5.3 Interactions between Traffic Filtering and Other Aspects of Networking

5.3.1 Routing. For a filtering tree or graph to work correctly, all packets destined for the protected target must pass through one or more forwarders or middleboxes acting as filters, in accordance with the intended design. This is the province of routing, which populates the forwarding tables used by forwarders. Routing is performed in several different ways—sometimes by a distributed algorithm that forwarders run among themselves, and sometimes by a centralized algorithm running in a separate controller.

Routing packets through a filtering tree may seem straightforward, but there is a different tree for each destination, and routing algorithms are also concerned with reachability, performance, fault-tolerance, and other policy constraints. For this reason, there has been considerable research on verifying that forwarding tables are correct, or on generating them correctly, where the correctness criteria include constraints about steering packets through filters [4, 10, 19, 24].

Another issue that complicates routing through a filtering tree is the fact that many traffic filters require session affinity—all the packets of a session, in both directions, must go through the same filter. Wide-area routing frequently creates different paths for packets traveling in different directions between the same two endpoints. Even packets traveling in the same direction may be spread across multiple paths because there has been a failure in one of the paths, or a need for better load-balancing. Within a cloud, where many virtual machines are running the same filtering software for scalability, a session can be assigned to any virtual machine. The assignment must be remembered, however, so that all packets of the session are steered in the right direction. Shortcuts such as "assign a session to one of four virtual machines based on the last two bits of some identifier" work well in static situations, but fail when filtering resources must be scaled up or down because of fluctuations in load.

5.3.2 Layering. Almost always, a packet arriving at a machine is being transmitted through multiple layered networks simultaneously, for example an Ethernet local area network, an IP network, and an application network. Figures 3 and 4 combined illustrate this simple example. In addition, layered between the application network and the lowest IP network there is often a virtual private network (Section 7.2.2) or other IP network. If the machine is actually a virtual machine in a multi-tenant cloud, there is sure to be at least one network between the tenant's IP network and the Ethernet, with the job of sharing cloud resources among all tenants.

The layers are significant because attacks can take place in any of them. This is both a challenge and an opportunity. If only packets in the lower layers are filtered, then many higher-level attacks will be concealed in the higher-level packets, which are mere data to the lower-level networks. For example, recall that IP intrusion detection systems look into packet contents for signs of malware at specific locations. These systems are assuming there are no networks layered between the filtering network and the application network; if there are additional networks, then the packet formats will be different, and the filtering criteria will be useless.

On the other hand, much can be gained by filtering packets separately in each network. Already there are special filters for Web requests and email messages, which are packets in their own application-oriented networks. These filters are deployed as middleboxes in these networks. This idea can be extended to intermediary layers, where each filter is attuned to the configuration, protocols, and vulnerabilities of its particular network. It is often possible to optimize architectures so that filters at multiple levels can be located on the same machines.

The second interaction between filtering and layering concerns networks below the filtering network in the layer hierarchy. Imagine that you have designed a filtering mechanism within a network, and convinced yourself that it is correct. Your argument concerns (among other things) paths in the network to a potential attack target, and shows that routing places an appropriate

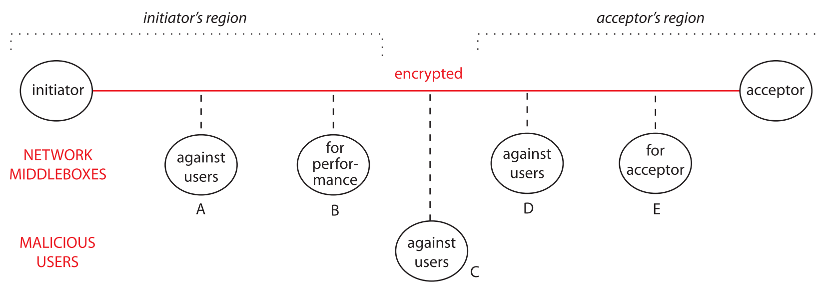

Fig. 12. A game: cryptographic protocols *versus* traffic filtering.

traffic filter in every path. Whether you remembered to state it or not, your argument that no (or a limited number of) attack packets reach the target depends on the assumption that attack packets do not suddenly appear *inside* the perimeter of traffic filters. It is easy enough to check that the network members inside the perimeter are part of the network infrastructure and therefore trusted, but what about the links? It must be ensured that no packet is received on trusted link that was not sent on the link. If the link is implemented rather than physical, it must be proved that the implementing network does not inject packets into the implementation of the link. This might seem like a fanciful concern, but it is not. An Ethernet can be penetrated physically, and it is easy to make a penetrated Ethernet inject packets into the links of networks layered on it [21]. In a multi-tenant cloud, the links of a tenant's network (where the filtering will take place) are virtual links implemented by sessions in the lower-level network that shares resources among tenants. If the cloud network does not isolate tenants properly, then packets sent by virtual machines of a different tenant could be delivered as part of this tenant's sessions.

5.3.3 Cryptographic Protocols. There is a profound interaction between cryptographic protocols and traffic filtering. If a user session is encrypted, then middleboxes in general, and traffic filters in particular, cannot read anything in the session packets beyond their headers. One response to this interaction is increased interest in filtering based on traffic attributes that encryption does not change, e.g., packet timing and sizes. It may be too early to tell how effective this will be as a defense, considering that its recognized successes are spying and tampering attacks [28].

In some cases, the relationship between users and filters is not adversarial, and there are three techniques for managing this interaction in more-or-less cooperative cases. Before presenting these techniques, we will explain the interested parties and their powers. We can think of their interactions as a game, one instance of which is illustrated by Figure 12. At the top of the figure we see what the initiating user can do. It chooses the acceptor of the session, and if the acceptor agrees, the data of the session will be encrypted end-to-end.

At the second level of the figure, we see what the network can do. The network has the power to insert middleboxes anywhere in the path of the user session, simply by routing session packets through them. The figure shows some common middleboxes, inserted in likely places, which are often in regions of the session path near the two endpoints. A middlebox might have the purpose of enhancing performance, for example by caching or compression (*B*). For maximum effectiveness, it should be placed near the initiator, as shown. A middlebox might be a traffic filter, with the purpose of protecting the acceptor from subversion attacks or policy violations that might damage it (*E*). This middlebox will probably be placed near the acceptor. Finally, the network might insert traffic filters that are working against the interests of the initiator and acceptor, either by preventing them from violating policies, or by spying on or tampering with their communication (*A* and *D*). These middleboxes might be placed in either region.

At the third level of the figure, we see that other malicious parties can also insert middleboxes in the path by various techniques such as wiretapping, for the purposes of spying and tampering (*C*). Fortunately, physical security and security mechanisms in other networks constrain such attacks. In the illustrated example, a malicious party is able to eavesdrop in the middle of the session path, but not near the endpoints.

If network middleboxes are working on behalf of the user endpoints of an encrypted session, and if they need to read data to do their work, then the cleanest arrangement is to make the middleboxes part of an application-oriented overlay network. This is illustrated by a SIP network in Figure 9. In the figure, data traveling on the links of the SIP network is encrypted by TLS in the IP networks, but each middlebox in the SIP network receives and sends plaintext.

The second and third techniques can be deployed within the bridged IP networks of the Internet. In the second technique, the network introduces a *proxy*—a middlebox that is a session-protocol endpoint—on behalf of the acceptor. The proxy accepts the initiator's TLS session (so it must have the server's identity and secret keys) and makes a TCP session between itself and the original acceptor. The proxy decrypts packets from the initiator and sends their contents in plaintext packets to the acceptor, so they can be read by any middleboxes in the path of the TCP session. In Figure 12, the acceptor would prefer a proxy between *D* and *E*, and the network would prefer it before *D*. The general idea of proxies that cooperate with endpoints is developed further in Middlebox TLS (mbTLS) [30].

The third and final technique aims to preserve both middlebox functionality and user privacy, based on new results in cryptography. At one extreme, fully homomorphic encryption [12] makes it possible to compute any function on encrypted data without learning more about the data than the function's value. Although fully homomorphic encryption is currently impractical (it is too expensive computationally, by orders of magnitude), there are less capable algorithms for computing functions on encrypted data with performance that may be feasible for current use [34].

6 DYNAMIC RESOURCE ALLOCATION

Because flooding attacks are resource wars, both network infrastructure and user members can defend against them by allocating more resources when they are under attack. Cloud computing has made it easier for networks to scale out traffic filters, and for users to scale out servers. Even in server-centric defenses the network is usually involved, for two reasons: (i) even when server resources are sufficient to absorb an attack, network bandwidth must also be sufficient to handle the attack, and (ii) the network provides the service of distributing the load across servers.

Dynamic resource allocation works better if resource replicas are geographically distributed, so that some replicas can be reached when other parts of the network are too congested. Because attacks on DNS servers are so common and damaging, it is especially important to have distributed authoritative DNS servers for popular domain names. Queries are distributed across the replicas by means of IP anycast. If there are five replicas sharing the load and one has been overwhelmed by an attack, IP anycast may not be dynamic enough to redirect queries away from the failed replica, but at least queries directed by anycast to the other four will succeed. In Figure 13, there are three authoritative DNS servers for the domain *example.com*; IP anycast directs the client's query to the closest one.

A "content-delivery network" provides many replicas of its customers' content, geographically distributed so that the latency of content delivery to each client is minimized. In Figure 13, the authoritative DNS servers for customer domain *example.com* are aware that its content is available at servers A through D, and also maintain information about location and recent performance of the servers. So each DNS server can return to a client the IP name of the best content server for it to contact.

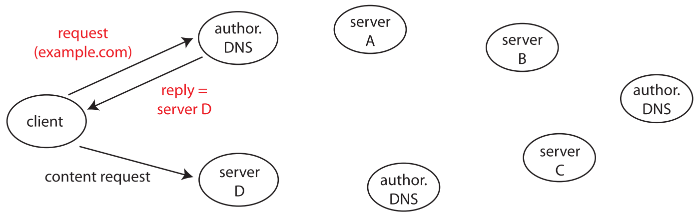

Fig. 13. Resource replication in a content-delivery network.

Replication of service resources is easiest when servers are responding to queries based on fairly static data. When queries can update service data, the service implementation must do extra work to keep the data replicas in some adequate state of consistency. (The study of distributed computing has produced many algorithms for replicated data, satisfying many different definitions of consistency.) In some cases dynamic data can be distributed across multiple sites more easily by sharding, e.g., by partitioning the keys of a key-value store so that each site is responsible for a subset of the keys. No one key-value pair will be replicated, but the total resources available will be greater.

Instead of dynamically allocating more server resources during attacks, the same result can be achieved by dynamically reducing the work per request that servers perform. For example, a flood of DNS queries is amplified when servers query other servers. A very effective defense against these attacks is longer times-to-live for cache entries, perhaps 30 minutes, in recursive and local DNS servers [27]. If local entries are cached longer, there will be fewer queries and retries made to authoritative servers. There are many good reasons for DNS cache entries with short times-to-live, but these can be changed as an adaptive measure during attacks.

SYN floods (Section 2.1.1) are such a serious problem that several specialized techniques have been developed for reducing server work per SYN, and these may be in use at all times rather than turned on just during attacks. In a "SYN cookie" defense, the server responds to a SYN with a SYN+ACK packet having a specially-coded initial sequence number (the cookie). It then discards the SYN, using no additional resources for it. If the SYN was an attack, it has caused little damage. If the SYN was legitimate, on the other hand, it will elicit an ACK from the initiator with the same initial sequence number incremented by one. By decoding the sequence number, the server can reconstruct the original SYN and then set up a real TCP connection.

Dynamic resource allocation is not much different from static resource allocation, so its interactions with other aspects of networking are already understood. Services under attack may be available in some places and not in others.

7 COMPOUND SESSIONS AND OVERLAYS FOR SECURITY

Like cryptographic protocols, compound session and overlays are employed by users to defend themselves against spying and tampering attacks (Section 2.1.4). Cryptography is not sufficient because it does not conceal packet headers.

Compound sessions and overlays are mechanisms through which users can insert their own middleboxes into session paths, and use them to conceal header information. Thus the entire topic of this section can be seen as an interaction between two patterns, namely compound sessions/overlays and traffic filtering.

7.1 Compound Sessions

7.1.1 Definition of Compound Sessions. A user member initiating a session to some far endpoint can insert another user member into the session path as a middlebox. To do this, the initiating user must give the name of the middlebox as the destination name of its outgoing packets, as shown in

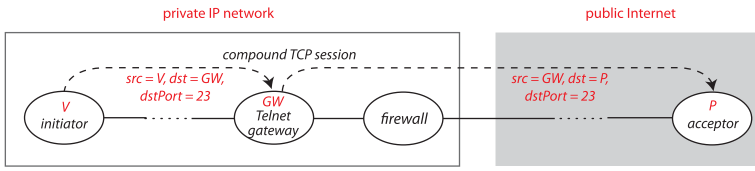

Fig. 14. A Telnet application gateway inside the access network of V.

Figure 14. The middlebox must learn the initiator's intended far endpoint, for example by getting it from some other field of the session-initiation packet. Then the middlebox changes the headers of the packets it receives (source becomes its own name, destination becomes the initiator's intended) and sends them out. Recall that a middlebox that behaves in this way is called a *proxy*. Each proxy accepts a session, initiates another session with a different header, remembers the association between the two sessions, and relays packets between them. A *compound session* is a chain of *simple sessions* composed by proxies in this way.

A compound session can have more than one proxy: the session-initiation packet can contain a list of proxies to visit, or a proxy can get the name of the next proxy or endpoint by using the session protocol to engage in a dialogue with the initiator. Because of the names in forward packet headers, return packets naturally pass through the same proxies in reverse order, and have their headers re-translated in reverse order.

The principal security significance of compound sessions is that each simple session has a different header, so compound sessions can be employed by users to obscure header information. In Figure 14, an observer between V and GW cannot observe the true acceptor of the compound session, at least from packet headers alone, and an observer between GW and P cannot observe the true initiator of the compound session.

Compound sessions are useful in many situations, but they have some limitations. After covering compound sessions, we will introduce overlays for security. These use explicit layering to create implicit compound sessions, and can do more for users than compound sessions alone.

7.1.2 Proxies in Access Networks. Perhaps the oldest example of a proxy for evading traffic filtering is an "application gateway," which is installed in a private IP network for the benign purpose of evading the too-simple filtering imposed by a firewall. For example, an enterprise firewall may block all outgoing sessions except Web accesses. However, the enterprise may also wish to allow outgoing sessions of another kind, when they are initiated by specific users. The firewall cannot enforce this policy because it does not know the mapping between internal IP names and users.

An application gateway for the application, for instance Telnet, solves this problem, as shown in Figure 14. To use it, a user initiates a Telnet session to the application gateway inside the enterprise network. The gateway is a Telnet proxy. By means of an extension to the Telnet protocol, which is embedded in TCP, the user supplies a password to authenticate himself to the gateway, and also the name of the real Telnet acceptor. The gateway initiates a Telnet session to the real acceptor outside the enterprise network, and joins the two simple sessions in a compound session. The enterprise firewall allows outgoing Telnet sessions from the application gateway only.

In this example, the operator of the enterprise network is cooperating with the user by providing the gateway. For the operator, it is easier and more efficient to provide the required user functions with an application gateway than with a greatly enhanced firewall.

In another cooperative situation, the operator of a private network might provide a proxy (with a public name) that initiators outside the network can connect to. The proxy authenticates the initiator as deserving the rights of members of the private network. Then, through the proxy, the initiator can connect to any member of the private network.

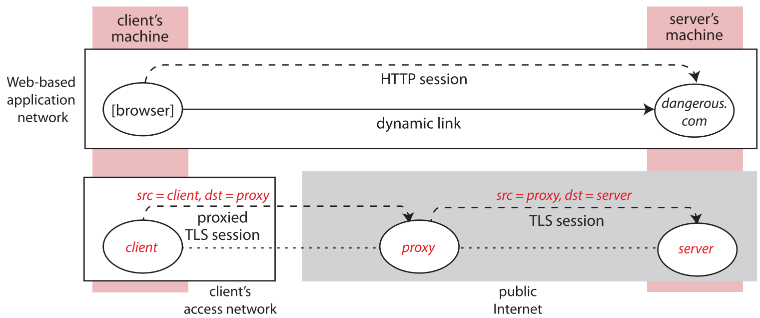

Fig. 15. A proxied TLS session protects the client's privacy in his access network, and provides anonymity at the Web server.

7.1.3 Proxies in Transit Networks. A user can evade filtering in his access network more easily by connecting to a friendly proxy in another network. This will be illustrated by the use of a proxy to reach a Web server.

In Figure 15, a secure dynamic link in a Web-based application network is implemented by a compound TLS session in bridged IP networks. First the browser's request causes initiation of a TLS session with a friendly proxy outside the client's access network. A proxied TLS session is like a normal TLS session except that: (i) instead of looking up the domain name *dangerous.com* and using its IP name as the destination of the session, the client's IP interface uses the proxy's IP name as the destination of the session; (ii) the client's IP interface expects and verifies the certificate of the proxy, not the Web site; (iii) the proxy decrypts the HTTP request in the TLS data, looks up the domain name, and uses the result of the lookup as the destination of an outgoing TLS session. After this the proxy relays packets between the two simple sessions of the compound TLS session (note that the proxy must decrypt and re-encrypt the data in each packet, because symmetric keys in the two simple sessions are different).

Because of the compound session formed by the proxy, the client's access network does not know what server the client is connected to, so the client has privacy from spying and tampering in his access network. The client also has anonymity at the server, because the server has no information about the client.

One disadvantage of this mechanism is that the client has no privacy from the proxy. Another disadvantage comes from the fact that the names of helpful proxies are usually publicly available (so users can find them), which means that they are available to the user's adversaries as well. Consequently, if the clients's access network is censoring the network activity of its users, it can simply block packets destined for external proxies. These disadvantages are addressed in subsequent sections.

7.1.4 Deflection. The problem that a censoring access network can block packets to known proxies has been addressed by several similar proposals [16, 18, 38]. They all use proxies, but in a way that still works despite the blocking.

A typical compound session in these proposals is shown in Figure 16. The access network of the session initiator is filtering out packets from users to certain destinations, represented here by the "covert destination." The initiator cannot evade this censorship by using a false source name, because then replies from the destination will not be delivered to the initiator (also, the network may be blocking *everyone's* access to the site). The critical mechanism is that session packets are routed through a friendly network where a forwarder recognizes that the packets must be treated specially, and deflects them to a proxy similar to the proxy in Figure 15.

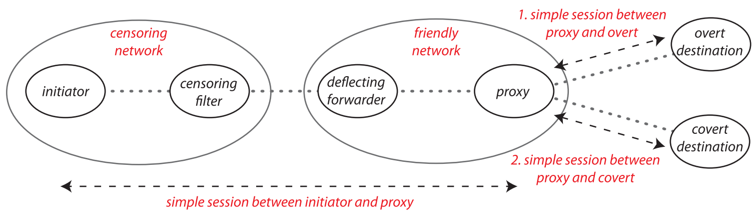

Fig. 16. A deflected session between an initiator and a covert destination. In the simple session on the left, names in the IP header are those of the initiator and overt destination; packets from the initiator are deflected to the proxy as an exception to normal forwarding.

For deflection to work, the initiator must give a hidden signal to the deflecting forwarder—one that the censoring network is unlikely to recognize—so the deflecting forwarder knows which packets to deflect. In Cirripede [16], the user registers with the friendly network; while the registration is active, all sessions initiated by the user are deflected.

In Figure 16, when the proxy first receives session packets, it initiates a TLS session to the overt destination. The TLS handshake is completed end-to-end between initiator and overt destination, so that all packets (including a certificate in plaintext) look normal to the censoring network. Once packet data can be encrypted, the proxy signals to the initiator that it is in the session path, terminates the session to the overt destination, gets the name of the covert destination from the initiator, initiates a session to the covert destination, and relays packets between the client and covert destination. During the entire compound session, the packets seen by the censoring filter will have the overt destination in their source or destination field.

The final problem to be solved is the placement of deflection forwarders in friendly networks. This can be viewed as a game between the censoring network (and its friends) and the session participants (and their friends). The administrator of the censoring network would like its outgoing packets to reach all or most of the public Internet without passing through a network with deflection forwarding. The Cirripede proposal favors deflection forwarders in networks close to the censoring network, so that many paths from the censoring network go through friendly networks, and the censoring network would suffer too much if it stopped bridging to friendly networks. The decoy routing proposal favors widespread deflection forwarders, in particular, in friendly networks close to a variety of important overt destinations. This way an initiator in the censored network can try several overt destinations until it finds one with deflection in the path, which it knows when the proxy signals its presence after the TLS handshake.

The rules of this game may change in the future: networks may be willing to give some path-selection control to cooperating networks and even user members, both of which are recommended by the SCION project [3]. Both now and in the future, when it comes to security contests, it matters who (and where) your friends are. Social forces will shape the Internet in their image, by defining its interest groups and alliances.

7.2 Overlays

An overlay is a virtual network layered on top of an underlay network (Section 3.3). We will first summarize the differences between overlays and compound sessions, then show their use in two security designs.

7.2.1 Overlays versus *Compound Sessions.* Figure 17 shows a prototypical overlay session whose links are implemented by sessions in one or more bridged underlay networks. All four machines are user members of their underlay networks. From the viewpoint of the underlay

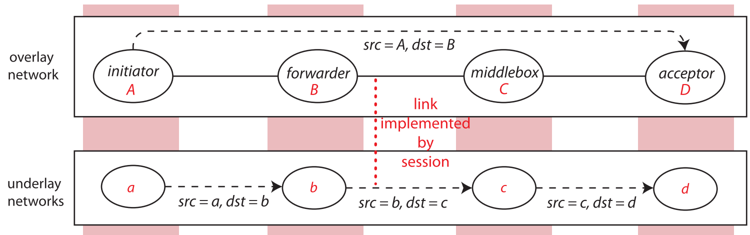

Fig. 17. A prototypical overlay session.

networks, this looks very similar to a compound session with three simple sessions connecting user members. Yet the sessions in the underlay are completely independent of one another (*b* and *c* are not proxies), and the overlay offers additional structures that are often useful, as follows:

- The overlay has its own namespace. Overlay names can be the same as in the underlays, but new names are useful for multiple purposes. For example, a member of a private IP network with a private, unreachable name can have a public, reachable name in an overlay.
- The overlay has its own routing. Overlay routing can insert application-specific middleboxes. In security designs, routing in an overlay is often used to vary and conceal packet paths.
- The overlay has its own session protocols. We'll see a good use of this in Section 7.2.3.
- The overlay has its own (geographical) span. It can unite allies in remote underlay networks.
- Sessions in the overlay and underlays have different durations. Overlay links—implemented by underlay sessions—are often long-lived and reused by many overlay sessions, which minimizes setup time and computational overhead (as in Section 4.4.2).

7.2.2 Virtual Private Networks. Strictly speaking, "virtual private networks" (VPNs) are not networks, but rather a technology for widening the geographic span of a private IP network such as an enterprise network. With VPN technology, an enterprise network is composed with other public and private IP networks in two ways simultaneously: (i) as usual, it is bridged with them, and (ii) it is layered on them, because some links of the enterprise network are implemented by sessions spanning other public and private IP networks. These relationships are illustrated by Figure 18.

In the figure, there is a TCP session between an enterprise machine and an employee laptop currently located in a coffee shop. The enterprise-network member on the laptop is described as a "VPN interface," because it is an IP interface plus VPN client. Before initiating the TCP session, it must first create a secure dynamic link to a VPN server in the enterprise network. To create the dynamic link, the laptop's VPN interface requests that its IP interface make an ESP session (Section 4.3) to public IP name *PS*. The employee must also enter a password to authenticate his identity to the VPN server. The ESP session happens to be compound, because it goes through a Network Address Translator (similar to a proxy) in the coffee shop's private IP network.

Viewed as an overlay network, the enterprise network uses VPN technology to allow a laptop in an insecure location to participate fully in the enterprise network. Most importantly, the VPN server assigns the laptop's member the name *V5* in the network's private namespace. This name can be chosen according to the privileges the laptop's owner has within the enterprise network. Consequently, traffic filters in the enterprise network can see from the source and destination fields of packets which policies should apply to the laptop's sessions, and enforce them accordingly.

7.2.3 Overlays for Anonymity. In Section 7.1.3, we showed how proxies in transit networks can provide session-initiating users with privacy within their access networks and anonymity at the

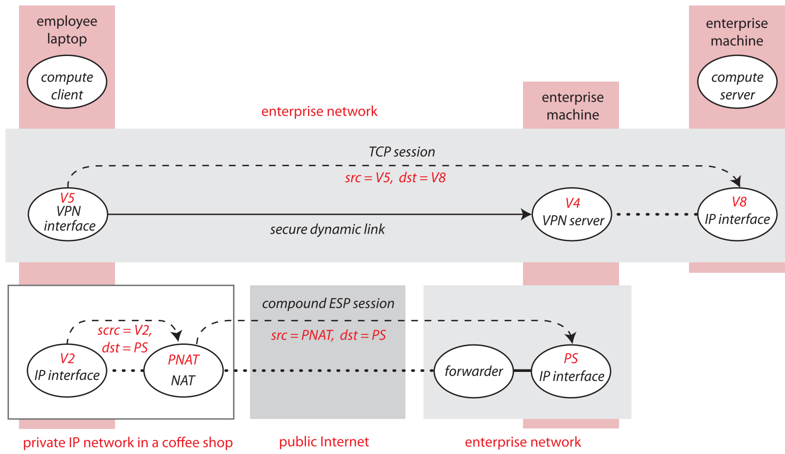

Fig. 18. An enterprise network using VPN technology. A secure dynamic link in the enterprise network is implemented by an ESP session in tunnel mode.

accepting endpoint. The weakness of this mechanism is that the user has no privacy whatsoever from the proxy. The purpose of the public service Tor [8, 31] is to add to the services above a high degree of privacy from the proxies.

Tor is an overlay network whose infrastructure members reside on the machines of volunteers world-wide. They are fully connected by long-lived links, each of which is implemented by a TLS session in the public Internet. This shows two of the ways Tor uses overlay properties: its membership unites allies across the globe, and its links are long-lived and reused by many overlay sessions (which minimizes setup time and computational overhead). An infrastructure member in Tor acts as a proxy within the overlay. Users also have Tor members on their machines. Each proxy has a public key, which it uses (with a certificate) to authenticate itself when setting up links by means of TLS sessions.

Tor is layered between application networks and the public Internet. Applications use the same interface to get TLS service from Tor as they would from the public Internet. User members query Tor directory servers to get lists of available proxies, each described by its public key, IP name, and policies.

To make a TLS session for an application (when there is no prior state in place), a Tor member first chooses a random route through several Tor proxies (this is why the proxies themselves do no routing). As with other overlay routing schemes, this varies and conceals packet paths. Next the user member creates a compound session in Tor that goes through the chosen proxies, as shown in Figure 19. The session protocol is the Tor "circuit" protocol, and each simple session is a Tor circuit with its own circuit identifier.

The important thing about circuits is that each one has a unique security association with the user member that created it. To make the compound session in Figure 19, the user first creates a simple session (circuit) to A, and executes a key-exchange protocol with A, so that each now knows a shared symmetric key K^{UA}. Next the user uses *circuit(UA)* to send to A an *extend* command telling it to create a new circuit to proxy B. Through the two associated circuits, U and B execute a key-exchange protocol, after which each has a shared key K^{UB}. Finally U tells B to *extend* the compound session by creating a new circuit to C, with U/C key exchange. Once a compound session has been assembled in Tor, it can be used to carry many TLS application sessions. In the

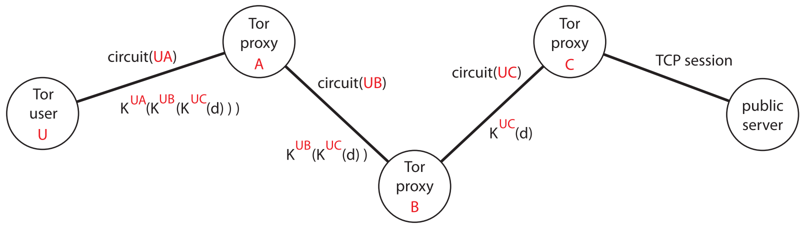

Fig. 19. A compound session made by Tor. The first three simple sessions use the Tor circuit protocol, and go through the Tor network. The last simple session uses TCP, and goes through the public Internet.

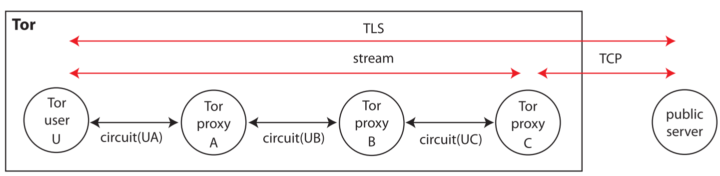

Fig. 20. Session protocols and their embeddings, for a single TLS application session made through Tor. Sessions of the circuit protocol last longer than application sessions.

background, the compound session is reconfigured piece-by-piece about once a minute, to confuse adversarial observers who are analyzing traffic patterns.[1]

Tor users use the security associations to conceal packet data from all except the last Tor proxy. The data transmitted on each circuit is multiply-encrypted as shown in the figure. When A receives a packet from U, it decrypts it before forwarding it to B, but it cannot read the packet because it is doubly encrypted with keys K^{UB} and K^{UC} that are unknown to A. For a similar reason, B cannot read it either.

To understand the rest of the Tor design, it is necessary to consider TLS as a separate protocol embedded in TCP. Tor has a second session protocol, the stream protocol, embedded in the circuit protocol. Figure 20 shows all the session protocols, with protocols above embedded in protocols below, used for a single TLS application session through Tor.

The stream protocol substitutes for TCP within Tor; there is a one-to-one correspondence between external TCP sessions and Tor streams, and TCP data is simply reformatted for streams. There are two reasons for using streams instead of TCP inside Tor: (i) if the data sent on Tor circuits were TCP packets, then proxy C would see their source and destination fields in plaintext; (ii) the reliable, ordered packet delivery of TCP is not required within Tor, because all of its links are implemented by TLS, and already have these properties.

When C has received enough stream packets to carry an HTTPS request with a domain name, it can complete the compound session end-to-end. It sends a TCP SYN packet with its own IP name as source and the IP name of the domain name as destination. After the TCP handshake, it continues converting data packets between the TCP and stream-protocol formats, and forwarding them in both directions. The TLS handshake between U and the server goes end-to-end, so that U can validate the server's certificate, and so that even C does not see plaintext packet data.

Unfortunately the Tor design for privacy has one serious deficiency, which is the fact that the final acceptor of the TCP session can know that Tor is being used, because there are readily

[1]Spies know that Tor sessions are deliberately concealed, so they have reason to analyze side-channels such as packet timing and sizes. These attacks can be successful in correlating packet streams coming into and out of Tor. Note that the adversarial observers can be Tor proxies, too.

accessible lists of Tor nodes. Fraudsters, spammers, and other criminals are big users of Tor, along with law-abiding people in need of privacy. Consequently an increasing number of services are rejecting or otherwise discriminating against Tor users [20]. Tor protects the reputation of its volunteer machines by allowing them to restrict their exiting TCP sessions or refuse to be exit proxies. Some volunteers must shoulder this burden, however, or the service will not be available to those who really need it.

8 CONCLUSION

Modeling and security are tightly intertwined. Given a rigorous model of a network, security attacks, and defenses, we can reason rigorously or even formally that the defenses will prevent the attacks—or at least mitigate them. Where there are gaps in the model, i.e., possible real-world behaviors that the model does not describe, there are possible attacks against which the defenses are useless.

As networks have become increasingly important in most aspects of daily life, their complexity has grown in proportion, and the early models have become increasingly inadequate. In this tutorial, a new model has enabled us to find a new and useful classification of security attacks, and to explain all common defenses by means of just four patterns. There is a clear relationship between the attack categories and the defense patterns, because the categories are based on which agents are the attackers, defenders, and potential victims, and some defenses are only available to some defenders. The model has also helped us understand how the patterns interact with each other and with other aspects of networking, which is a dimension of security that has received little prior attention. The modeling and defenses in this tutorial are obviously not complete, yet we believe that any progress toward organized thinking about network security will be helpful for building stronger defenses.

REFERENCES

[1] Arbor Networks. [n.d.]. NETSCOUT Arbor's 13th Annual Worldwide Infrastructure Security Report. https://pages.arbornetworks.com/rs/082-KNA-087/images/13th_Worldwide_Infrastructure_Security_Report.pdf.

[2] Katerina Argyraki and David R. Cheriton. 2005. Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX ATC*.

[3] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. 2017. The SCION Internet architecture. *CACM* 60, 6 (June 2017), 56–65.

[4] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *Proceedings of ACM SIGCOMM*.

[5] Ran Canetti. 2019. Universally Composable Security: A New Paradigm for Cryptographic Protocols. https://eprint.iacr.org/2000/067.pdf.

[6] David D. Clark. 1988. The design philosophy of the DARPA Internet protocols. In *Proceedings of SIGCOMM*. ACM.

[7] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. 2005. Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking* 13, 3 (June 2005), 462–475.

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*.

[9] Dyn [n.d.]. Dyn Analysis Summary of Friday October 21 Attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[10] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*.

[11] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the ACM Conference on Computer and Communications Security*.

[12] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nockolai Zeldovich. 2013. Reusable garbled circuits and succinct functional encryption. In *Proceedings of Symposium on Theory of Computing*. ACM.

[13] Tyrone Grandison and Morris Sloman. 2000. A survey of trust in Internet applications. *IEEE Communications Surveys and Tutorials* 3, 4 (2000), 2–16.

[14] Mark Handley. 2006. Why the Internet only just works. *BT Technology Journal* 24, 3 (July 2006), 119–129.

[15] Mark Handley, Vern Paxson, and Christian Kreibich. 2001. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*.

[16] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the ACM Conference on Computer and Communications Security*.

[17] ITU. 1994. *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. ITU-T Recommendation X.200.

[18] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. 2011. Decoy routing: Toward unblockable Internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*. USENIX.

[19] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real time network policy checking using Header Space Analysis. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*.

[20] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Vern Paxson, Steven J. Murdoch, and Damon McCoy. 2016. Do you see what I see? Differential treatment of anonymous users. In *Proceedings of the Network and Distributed Security Symposium*. Internet Society.

[21] Timo Kiravuo, Mikko Sarela, and Jukka Manner. 2013. A survey of Ethernet LAN security. *IEEE Communications Surveys & Tutorials* 15, 3 (2013), 1477–1491.

[22] Adam Langley et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of ACM SIGCOMM*.

[23] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security* 6, 1 (February 2003), 128–171.

[24] Nuno P. Lopes, Nikolaj Bjorner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking beliefs in dynamic networks. In *NSDI*.

[25] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. 2002. Controlling high bandwidth aggregates in the network. *Computer Communication Review* 32, 3 (July 2002), 62–73.

[26] Catherine Meadows. 1996. The NRL protocol analyzer: An overview. *Journal of Logic Programming* 26, 2 (February 1996), 113–131.

[27] Giovane C. M. Moura, John Heidemann, Moritz Muller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the ACM Internet Measurement Conference*.

[28] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the ACM Conference on Computer and Communications Security*.

[29] David Naylor, Alessandra Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafo, Konstantina Papagiannaki, and Peter Steenkiste. 2014. The cost of the 'S' in HTTPS. In *Proceedings of ACM CoNEXT*.

[30] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. 2017. And then there were more: Secure communication for more than two parties. In *Proceedings of ACM CoNEXT*.

[31] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. 1998. Anonymous connections and onion routing. *IEEE JSAC* 16, 4 (May 1998), 482–494.

[32] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. 2000. Practical network support for IP traceback. In *Proceedings of SIGCOMM*. ACM.

[33] Y. Sheffer, R. Holz, and P. Saint-Andre. 2015. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). *Internet Engineering Task Force Request for Comments 7457*.

[34] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. BlindBox: Deep packet inspection over encrypted traffic. In *Proceedings of SIGCOMM*.

[35] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[36] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2018. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials* 20, 1 (2018), 465–488.

[37] Janet Vertesi. [n.d.]. My Experiment Opting Out of Big Data Made Me Look Like a Criminal. https://time.com/83200/privacy-internet-big-data-opt-out.

[38] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the network infrastructure. In *Proceedings of the USENIX Security Symposium*.

[39] Abraham Yaar, Adrian Perrig, and Dawn Song. 2003. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[40] Pamela Zave and Jennifer Rexford. 2019. The compositional architecture of the Internet. *Commun. ACM* 62, 3 (March 2019), 78–87.

[41] Pamela Zave and Jennifer Rexford. 2019. Patterns and Interactions in Network Security. *arXiv 1912.13371 [cs:NI]*.

Received January 2020; revised June 2020; accepted July 2020