Patterns and Interactions in Network Security

Pamela Zave and Jennifer Rexford

April 2, 2019

1 Introduction

This article is intended as a concise tutorial on a very large subject. Network security encompasses both the security of networks themselves, and the security properties expected by network users as they entrust their data communications to networks.

The focus of this tutorial is derived 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 rather than the individual details, 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. Although these interactions are not frequently discussed, they 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 §2 defines network security in two ways. First, we classify security threats based on the way that network engineers might see them, with awareness of the defenses available against various threats. Second we discuss how network security is related to other forms of cyber-security, and the gaps where no comprehensive defenses yet exist.

Next, §3 elaborates on how we describe networks and network services. We use a new model emphasizing that 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 a namespace, routing, forwarding, session protocols, and directories). This model allows complete and precise descriptions of today's network architectures. It also encourages recognition of reusable patterns: in principle, any security measure might be found in any network in a compositional architecture.

The main three sections of the tutorial cover the three major patterns for providing network security, in the most convenient order for exposition. Within

a single network, speaking at the very highest level of description, networks protect themselves and their users by packet filtering (§5). Users protect themselves with cryptographic protocols (§4), which hide the data contents of packets. Cryptography cannot hide packet headers, however, because the network needs them to deliver the packets. So when users need to hide packet headers from adversaries, which may include the network from which they are receiving service, they must resort to compound sessions and overlays (§6). The first two patterns will be familiar to anyone who has even dabbled in network security, while the importance of the third pattern has not been sufficiently recognized.

In addition to explaining the basics of network security, we will consider how security mechanisms interact with other mechanisms within their networks and across composed networks. Among other goals, this helps determine where security could and should be placed in a compositional network architecture. Each of the three main sections includes a discussion of interactions.

Currently interest in verification of trustworthy network services is increasing rapidly. We hope that this tutorial will convince the reader that network security cannot be understood without considering the compositional nature of network architecture. On the positive side, a compositional view helps identify reusable patterns subject to reusable implementation and verification. The tutorial concludes (§7) with a brief assessment of prospects for holistic verification of network security.

2 What is network security?

2.1 Background

§3 will present in some detail how networks and network services can be described rigorously for studying network security. In the meantime, this section explains a few basic concepts necessary to understand the threats.

A *member* of a network is a software or hardware module running on a computing *machine,* and participating in the network. The member can send or receive digital *packets* on one or more links of the network, and implements some subset of the network's protocols. The member is the machine's *interface* to that network.

A network member usually has a unique *name* in the namespace of the network. A network defines a format for transmitted packets, each of which has a *header part* and a payload or *data part.* Each header usually includes a *source name* indicating which member originally sent the packet, and a *destination name* indicating which member is intended to ultimately receive the packet.

In our model a network always has a single *administrative authority (AA)*, which is a person or organization responsible for the network. The AA provides and administers resources for the network, in the form of links, members, and additional resources on the members' machines. The AA 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 AA to provide services, and *user members* belonging to the network for the purpose of employing its services.

There are many kinds of network, ranging from the physical (concrete) to the virtual (abstract). As we shall see in §3, a machine usually participates in several networks simultaneously. A distributed application system can be viewed as a network, even though its structure as a network may be uninteresting, and its AA may be a loose organization of peers. Consider a machine participating in a distributed application system whose members are connected by the Internet, which is an association of networks all using the Internet Protocol (IP) suite. In our model, the machine has members of (interfaces to) both the application system and the Internet. These members communicate through the operating system of their machine. When an application member sends a packet, the application packet will be passed from the application member to the co-located Internet member. The Internet member will encapsulate the entire application packet (header and data) in the data part of an Internet packet, prefix an Internet header to it, and send it through the Internet.

2.2 A practical classification of threats to network security

Network security is a pragmatic subject with boundaries that have been drawn by convention over time. In §2.3 we will talk about the boundaries between network security and other kinds of cyber-security, and the gaps where no comprehensive defenses yet exist.

It seems to be a hopeless task to classify security attacks directly—by their very nature, they are crafty, they exploit gaps in standard models, and they are always evolving. Defenses, on the other hand, fit into well-understood patterns. For this reason, our classification of security threats is based primarily on the factors that cause particular defenses to be applicable. This classification is not an exact partition of security threats, because: (i) there are overlaps between categories because multiple defenses may apply, (ii) inevitably, it does not cover all security attacks, and (iii) it ignores the fact that some "attacks" are mere mistakes.

2.2.1 Flooding attacks

A *resource attack* seeks to make its victim unavailable by exhausting its resources. In networking, resource attacks are usually called *flooding attacks,* because they entail sending floods of packets toward the victim. Flooding attacks are one type of *denial-of-service* attack.

The intended victims of flooding attacks vary. If the victim is a public server or user member, the attack might seek to exhaust its compute-cycle or memory resources, or the bandwidth of its interfaces to the physical world. 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. 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.

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! For this reason, an effective flooding attack always employs 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 Internet members, 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. A flooding attack from many network members, particularly members of a botnet, is called a "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. In a Web-based application network, the attacker can send particular HTTP requests that force a Web server to do a large amount of computation.

- 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).

- In an Ethernet network, a forwarder's response to receiving a packet to an unknown destination is to "flood" the network with it, which means (in this case) to send it out all links in the "spanning tree" so that eventually all members receive it and the designated destination responds to it. An attacker can amplify any packet this way, simply by putting in an unused destination name.

- Email spam and voice-over-IP robocalls can exhaust the capacity of a machine's interface to the physical world, which in these application networks depends on the time and patience of people.

Network infrastructure provides the principal defense against flooding attacks, by filtering out attack packets (§5). Flooding attacks can also be countered by allocating additional resources to handle peak loads (also §5); 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, none of the bots sending attack traffic are actually responsible for the attack. Even if bots use their true source names, there may be too many of them to cut off.

- 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.

The examples of amplifications and hiding techniques show that flooding attacks are network-dependent, because they exploit vulnerabilities in the protocols of specific networks. Nevertheless, their effects are *not* network-dependent, because of "fate sharing." All the network members and applications on a machine share the same physical resources and physical network links, so if resource exhaustion causes a machine to crash, thrash, or become disconnected, all programs running on the machine will share the same fate.

Flooding attacks are a very serious problem in today's Internet. There are businesses that generate them for small fees. They target popular Web sites and (especially) DNS [15]. 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.

2.2.2 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 of subversion attacks:

- The attacker sends malware to infect or penetrate the machine. The malware might be spyware, ransomware, or turn the machine into a bot. The malware might exploit the machine's resources, steal or damage data stored in the machine, or attack the physical world through devices controlled by the machine.

- Port scanning is the process of trying every TCP or UDP destination port on an IP endpoint, to see if it will accept a session initiation. Port scanning does not in itself do much harm, but should be prevented because it is gathering information to be used in launching other malware attacks. This is because most malware targets a known vulnerability in a specific program or application.

- An attacker can give an infrastructure member of a network false information. The best-known of these attacks is "BGP hijacking." BGP is the control protocol through which IP networks exchange routing information. In a hijacking attack, an attacker tells an IP router to send packets with certain destination names to the attacker. If the attack succeeds, all packets sent to the router with those destination names will be forwarded along a path to the attacker rather than along a path to their true destinations. Although the attacker may do nothing with the packets but drop them, it can now practice subversion by impersonating the intended destination, possibly stealing commerce or secrets. Subversion attacks on DNS inject false entries into the directory; they can make services unavailable to users, or allow the attacker to impersonate the intended destination.

As with flooding attacks, network infrastructure protects itself and its users from subversion attacks by packet filtering. But subversion attacks are significantly different from flooding attacks because they often require two-way communication between attacker and victim. This means that an attacker cannot hide by spoofing. If the attacker put a false source name in its first packet to a victim, it would never receive a response and could not complete the attack.

2.2.3 Policy violations

Obviously, the default behavior of a network is to provide all communication services requested of it. These services should be provided according to explicit or implicit agreements about quality and privacy.

The final two categories are almost duals of one another. In both cases, there is an exception to the default behavior, and the network infrastructure attempts to tamper with a specific communication (up to and including blocking it) or spy on it.

In the category of policy violations, we have specific communications that laws, business agreements, organizational practices, or social rules say should not occur, at least not without interference. We support the efforts of the network infrastructure to block it, record it, tamper with it, or rate-limit it. The next category (§2.2.4) is in some cases exactly the same, except that we are supporting the network users in trying to avoid interference with their communications.

Examples of policy violations include:

- Two users can willingly participate in illegal communication. 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.

- A minor can attempt to access a Web site that violates parental controls, which should be prevented.
- 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.

As with flooding and subversion attacks, network infrastructure defends against policy violations by packet filtering. Policy violations are significantly different because they are so specific—they usually involve at least one specified individual member. This distinguishes them from flooding and subversion attacks, whose origins are usually unknown, and whose targets are often opportunistic.

Other factors that distinguish policy violations from flooding or subversion attacks are (i) the victim of policy violations is not usually a recipient of packets, but whoever made the policy being violated (such as parents wanting control over their children's Internet usage); (ii) not all policy violations are blocked, which is always the most desirable response to flooding and subversion attacks.

2.2.4 Spying and tampering

§2.2.3 introduced the relationship between these two categories. 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 are often in the network infrastructure itself. Social debates involving legal, ethical, political, and commercial considerations should not be constrained by technical considerations. The goal of technical experts should be to have the knowledge to implement whatever decisions emerge from these debates [13].

The reason that spying and tampering is not the exact dual of policy violations is that policy violations are always countered by network infrastructure on behalf of the network's AA, while spying and tampering can be carried out both by network infrastructure and by other attackers (see §3.1).

For some purposes, spying requires reading the data parts of transmitted packets. For other purposes, it is sufficient for the attacker to observe packet headers, sizes, and timing. 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 as a tool in repression of or retaliation against political dissidents.
- By monitoring the searches and Web accesses of a network user, an attacker can learn a great deal about the user's personal life.
- Network infrastructure 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 session protocols (§4). The second is the use of compound sessions and overlays (§6).

2.3 Relation to other kinds of security

For users, network security is a first line of defense against subversion attacks; a major goal is to to keep subversion packets from being delivered to user machines. If the packets do arrive, however, then security measures in operating systems and applications must take over. Many applications and all operating systems have well-developed security measures of their own.

There is a large body of work on "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, [19, 33]). Trust management is a decision-making component of most forms of security, including network security. Distributed trust-management systems also rely on network security, for instance to communicate secret information safely among nodes of the system.

Personal data privacy is a form of security that is much discussed in today's world. People 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. Network privacy—privacy about one's usage of a network—can contribute to personal data privacy, but only in a limited way. For example, privacy from network spying can enable people to access search engines and read Web sites anonymously, at the cost of longer delays and worse search results (because they are not customized). On the other hand, people cannot participate in social media or electronic commerce in any meaningful way while preserving anonymity.

One of the most challenging aspects of the drive to protect privacy, including both privacy of personal data and privacy from network spying, is the problem of "covert channels" or "information leakage." Attackers can observe many things in addition to what is intended or explicit, including timing, usage of shared

resources, configuration details that distinguish one "identical" machine or software copy from another (called "device fingerprinting"), and electromagnetic radiation across the entire spectrum. Attackers can correlate both covert and overt information from diverse sources, including simultaneous observations of different parts of a network. By applying both statistical and logical analysis to all this data, attackers can reach surprisingly precise conclusions. This is an area in which attackers appear to be well ahead of defenders, at least in principle. The only consolation is that covert channels leak information to attackers at very low bandwidths compared to the overt network channels protected by network security.

Another area in which attackers appear to be ahead of defenders, again due to the heterogeneous and unpredictable nature of the threats, is attacks by social engineering. Such attacks include phishing and guessing passwords. Note that botnets are heavily populated by Internet of Things devices such as baby monitors, because they come with factory-installed default passwords, and their naive owners do not change their passwords. Social attacks also include insider attacks, where security software has bugs or backdoors installed by employees with access to code.

3 A model of networking

3.1 Network links

A network has *links,* which are communication channels on which digital packets can be sent and received. Most physical links are wires, optical fibers, radio waves, or microwaves. To be a member of a network, a hardware or software module on a machine must be able to send or receive on one or more links of the network.

Network security uses many forms of digital technology to transform physical networks into virtual networks with more secure behavior. Consider, for example, a wireless (radio) network. It has a single many-to-many link on which any machine with a transmitter within radio range can send packets; similarly, any machine with a receiver within radio range can receive any packet being sent. In other words, anybody's machine within radio range can have a member of this network. If the network uses secure WiFi protocols, in contrast, packets "on the air" are encrypted. In a wireless network using these protocols, only authorized members have the keys to encrypt and decrypt packets, so only authorized members can send and receive in any meaningful sense.

In wired networks, buses (used in older Ethernets and cable networks) are also many-to-many links, so packets can be sent and received by any machine connected to them. Even in a network with wires for (supposedly) point-to-point links, a machine tapped into a wire can send and receive packets on the wire. Even without wiretapping, a wired link can be compromised by weak physical security. For example, it may be assumed that the endpoints of a wire are plugged into known machines because the wire is in a private building with

physical security, yet this assumption will be false if an unsupervised visitor plugs his laptop into an unused wall port, or moves a wire from a desktop machine to his laptop.

3.2 Composition of networks

As mentioned in §2.1, there are many kinds of network used for many purposes, and this tutorial should make sense for all of them. The principle example to be covered, however, is the Internet. The Internet consists of a large number of IP networks (networks using the Internet Protocol suite), so the constituent networks may differ in their AAs, but not their basic design. These networks are connected together at various points by *bridging,* which means that two particular networks share some links so they can forward packets to each other.

An IP network in the Internet is typically classified as either public or private. A public network accepts user members without authorization, so any machine can have a member of a public network, and the network cannot trust its user members. A private network only allows authorized user members, so private networks can assume that members are trustworthy, at least to some extent. Recalling the unsupervised visitor above, it is worth noting that authorization of members can take many forms, and assumptions based on it should not be made casually.

A network provides one or more *communication services.* A particular instance or usage of a communication service is called a *session.* A communication service is usually associated with a *session protocol,* which is the set of rules governing packet formats and sender/receiver behavior during sessions of the service. Like a link, a session is also a communication channel for a group of digital packets.

In addition to being composed by bridging, networks are often composed by *layering.* Formally, a network (the *overlay* relative to composition) is *layered on* another network (the *underlay*) when a *link* of the overlay is implemented by a *session* of the underlay. Since the implementation always consists of digital logic, whether hardware or software, an overlay link is always virtual, regardless of whether the links in the underlay are physical or virtual. For example, Figure 1 shows how an IP network in the Internet may be layered on several local-area networks.

In the figure, shaded boxes are machines with members of (interfaces to) multiple networks. In the IP network, A is the IP address (name in the namespace of the IP network) of the member on Alice's machine, while B is the IP address of the member on Bob's machine. These members are currently the endpoints of a TCP session. Packets on a path of links between A and B are forwarded by IP routers named $R1$ and $R2$.

On the lower level of the figure there are three isolated local-area networks. In the local-area networks, names are Ethernet addresses; we show the Ethernet name of a member simply as the lower-case version of the IP address on the same machine. Each IP link is implemented by an Ethernet session, as indicated by the bold arrows. As mentioned in §2.1, the actual mechanism is that members

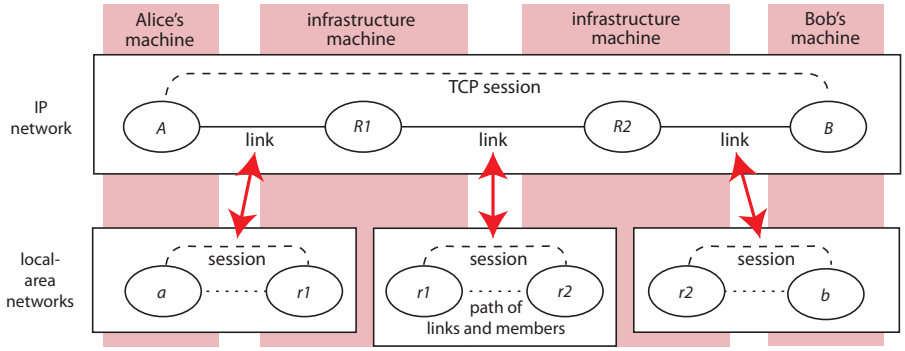

Figure 1: The IP networks of the Internet are layered on many local-area networks.

of different networks on the same machine communicate through the operating system and/or hardware of the machine, and IP packets on the link are actually transported by local-area networks as the data parts of Ethernet packets. There is no bridging in Figure 1; rather, the IP network spans multiple local-area networks by forwarding on paths that concatenate the links they implement.

This definition of layering is very different from the older notion of layering in networks found in the "classic" Internet architecture [11] and OSI reference model [25]. In the older meaning, each layer in a network has a distinct function, and the number of layers in an architecture is fixed. In the new meaning, each layer is a self-contained network with the potential to include all the basic structures and mechanisms of networking. Each network may be specialized with respect to its purpose, membership scope, geographical span, and role in the layer hierarchy. An architecture is a flexible composition of as many networks as needed. Motivations and further explanations of the new compositional model are given in [51]. We use it in this tutorial because it allows comprehensive yet precise descriptions of how networks in general, and the Internet in particular, actually work today.

3.3 Properties of channels

In this tutorial the default communication channel (whether link or session) is two-way and point-to-point, meaning that it has two *authorized* endpoints, each one capable of sending and receiving packets. (In principle, unless proven otherwise, it can also have unauthorized senders and receivers.) Even in networks with broadcast physical media, layering of virtual networks over the physical network quickly transforms the single many-to-many link into many one-to-one links. For performance and fault-tolerance, links have measurable properties such as packet-loss rate, bandwidth, latency, and jitter.

Enforcing logical properties of channels is a critical part of network security.

In this section we distinguish two very different ways of doing this.

A one-way point-to-point link or session has the logical property of *data integrity* if: a packet is received by the receiving endpoint only if it was sent by the sending endpoint. (The extension to two-way channels is obvious.) Attacks—and also normal network transmission—can duplicate packets, so it is necessary to add that if the sender sends a distinguished packet m times, the receiver can receive it at most m times.

A one-way point-to-point link or session has the logical property of *data confidentiality* if only the receiver can read the data sent by the sender. (The extension to two-way channels is obvious.) Confidentiality should include "forward secrecy," which means that the data remains secret in the future, even if attackers save encrypted packets, later learn current secrets of one of the endpoints, and try to use the current secrets to decrypt old packets.

A "secure" link or session has both data integrity and confidentiality. These properties must hold despite the possibility of physical attacks as in §3.1. They must also hold despite the fact that a session can be attacked by any untrustworthy network member in the path of session packets, and a virtual link can be attacked by any attack whatsoever on the session implementing it. It is accepted that observers will have access to packet headers, sizes, and timing.

The first way to establish logical properties of channels is through the use of a session protocol. In particular, the security properties of data integrity and confidentiality can be enforced by cryptographic session protocols (§4). It is important that cryptographic protocols are proved correct even in hostile environments. For example, in accepted proof systems such as the NRL protocol analyzer [37] and Universally Composable Security [9], the baseline model of a security protocol allows an adversary to control all communication channels among participating agents, reading, deleting, injecting, or altering any packets that the adversary wishes. If a cryptographic session protocol is proven correct with such conservative assumptions, the endpoints of a session using it can trust that the session has data integrity and confidentiality even without the cooperation of other agents.

The second way to establish logical properties of channels is through bottom-up reasoning. To reason that a physical link has no senders or receivers except the authorized ones, it is necessary to talk about physical security. A path of network members and secure physical links can be established as secure by reasoning about the network members. Established properties of a path can be used to guarantee properties of a session whose packets traverse the path; established properties of a session can be used to guarantee properties of a link implemented by the sesssion.

4 Cryptographic protocols

Cryptographic protocols are incorporated into the session protocols of a network. They are most commonly employed by user members of a network, to ensure that a point-to-point session between them has the properties of data integrity

and confidentiality. The members can also achieve *endpoint authentication*, which means that either session endpoint can be sure of the other endpoint's identity. Cryptographic protocols protect users against spying and tampering attacks. When the protocols are employed by infrastructure members, they protect against subversion of the network infrastructure, as well as spying and tampering.

§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 (§4.2), because it provides some crucial functions and supports others. In §4.3 we return to the properties of data integrity and confidentiality. Finally, in §4.4 we discuss architectural interactions with cryptographic protocols that are relatively independent of other security patterns.

In §4.1 through §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. §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 given to the other endpoint of the session in answer to the question, "Whom am I talking to?"

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.

To understand where identities come from, consider the Web session pictured in Figure 2. At the lower level, a TCP session initiated by C and accepted by S traverses a chain of bridged IP networks. At the upper level, a distributed system is viewed as a network, which is always possible even though their structures as networks are usually too simple to bother with. In this Web-based application network there is a dynamic link (implemented by the TCP session) between a browser and a server, on which an HTTP session is taking place. Placed above the client's browser there is a user whose clicks and keystrokes provide input to the browser.

In this example, the server's machine has two interfaces to two network members, each with a name in the namespace of its network. In its IP network it has IP address S. In the Web network it has domain name *bigbank.com*. The client's machine also has two network members, but the browser does not really have a name in the Web-based application network, because it only initiates

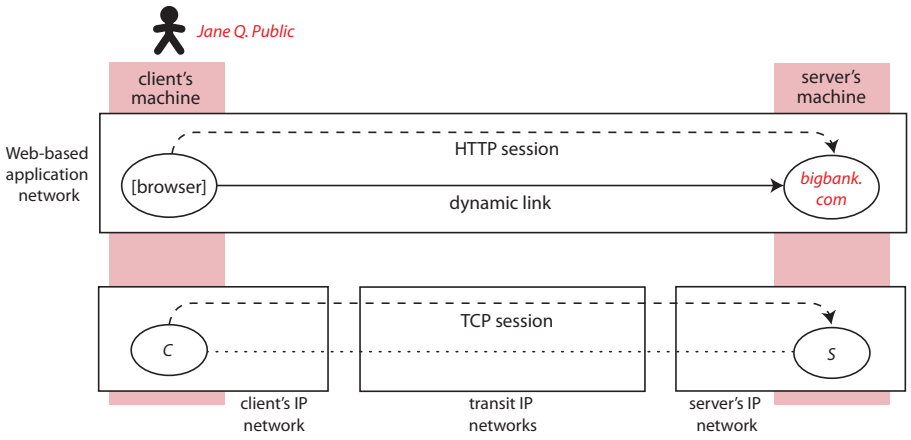

Figure 2: Member names and identities in a Web application.

sessions and never accepts them. However, the user of the browser is a person named *Jane Q. Public*, and we can imagine her as a member of an even-higher-level distributed financial system.

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 network 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 with too little meaning in the outside world. In practice the server's IP interface S would be known by its Web name *bigbank.com*, and the browser's IP interface C would send its user's name *Jane Q. Public*.

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 (semantics given in the next section). The relationships among the important entities are shown in Figure 3. 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.

It should be noted that trust between communicating endpoints is not nec-

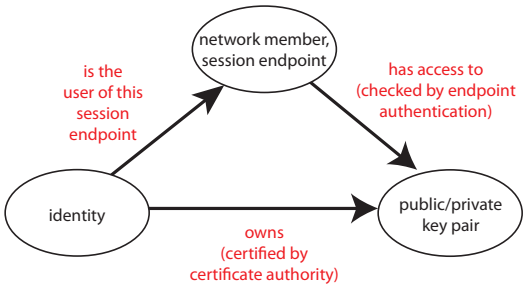

Figure 3: Relationships among identification entities.

essarily simple or absolute. For instance, two endpoints may be communicating to negotiate a contract, and (because they do not trust each other completely) need to communicate through a third party trusted by both. A trusted broker can ensure, for example, that both parties sign the exact same contract [8].

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 2048 bits to be considered secure (the minimum size is expected to increase in the future). A public key is a pair of unsigned integers (n, e). The corresponding private key is a pair (n, d). To be encrypted, a message must be divided into chunks such that each chunk has an integer representation less than n. If m is such a chunk, then the public-key encryption of m is $m^e \bmod n$, and the private-key encryption of m is $m^d \bmod n$. The point of this isolated detail is to show why public-key cryptography is computationally expensive, which is an important factor in design of cryptographic protocols. For a readable technical account of public-key cryptography, see Kurose and Ross [32].

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 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. Certificates are often validated poorly or not at all, which is why some client software has been dubbed "the most dangerous code in the world" [17].

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.

As mentioned in §4.1, IP names (addresses) are not very good identities, because they are often assigned transiently, and are never mnemonic. As a result, the names of IP members cannot be authenticated, leading directly to the prominence of spoofing in a variety of security attacks. The Accountable Internet [2] is a proposal based on the alternative principle that Internet names should be the persistent identities of Internet members, and that they should be "self-certifying." This means that any other member communicating with a member can authenticate its name, even without trusting a certificate authority. This is important in a global network, because there are no certificate authorities that are trusted by all countries [12].

Clearly this could be achieved if the name of a member were its public key, but public keys are too long for network names. The Accountable Internet solves this problem by using as a member's name a 144-bit *cryptographic hash* of its public key. A cryptographic hash is computed by a function H 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 $H(m) = H(m')$. In AIP, having validated that a member has public key K^+, a validator completes the job by computing $H(K^+)$ and checking that it is the same as the member's name.

In the Accountable Internet Protocol (AIP), endpoint authentication is not implemented in user endpoints by session protocols, as is usual; rather it is part of routing and forwarding, and is implemented in AIP forwarders. The costs are considerable and everyone connected to the Internet must bear them, which is why AIP is a radical proposal. The Accountable Internet's counter-argument would be that endpoint authentication is essential for network security, so everyone needs it all the time.

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 encrypts 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. In practice a (short) cryptographic hash $H(m)$ of the document is 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 both computed values are the same.

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. This kind of delegation is used in Named Data Networking [52].

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 of key exchange is the Diffie-Hellman algorithm, shown in Figure 4.

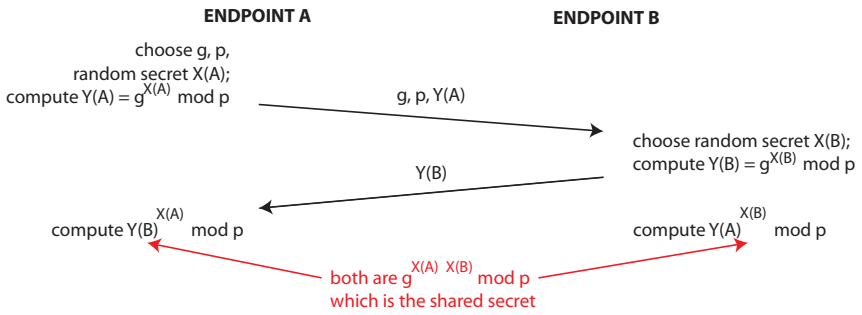

Figure 4: Diffie-Hellman key exchange. g is a small number such as 2 or 3, while p, $X(A)$, and $X(B)$ are large integers.

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 5 shows how such an attack would work. 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.

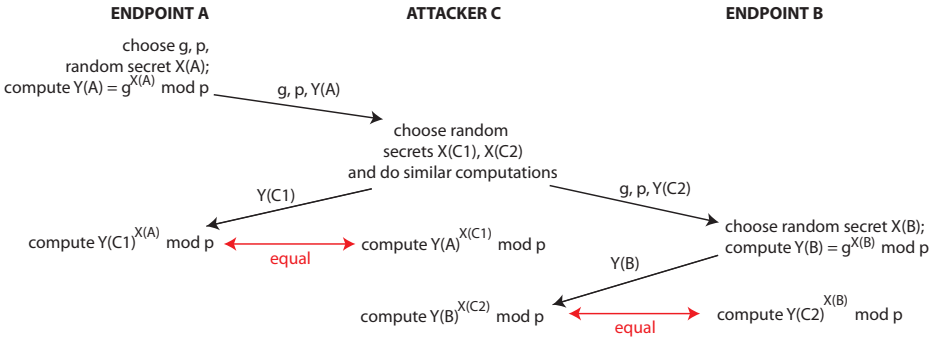

Figure 5: A man-in-the-middle attack on Diffie-Hellman key exchange.

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, the message of the protocol sent from each to the other 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 Two IP cryptographic protocols

This section provides an overview of the two most important cryptographic protocols in the IP protocol suite, TLS and ESP. Transport Layer Security (TLS) is the successor to Secure Sockets Layer, and is an extension of TCP. "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, and data confidentiality. They have interesting differences, and the differences are significant for their use in compositional network architectures. To explain these differences, we must begin with an explanation of how session protocols are composed.

4.3.1 Composition of session protocols

A *message* is a semantic entity within a session protocol. Because of the conversational nature of protocols, it would be unusual for a packet to contain multiple messages, but length restrictions could easily cause a message to be transmitted in multiple packets. *Control messages* are used by a protocol to synchronize the endpoints and share specific parameters. *Data messages* contain the substance being communicated. Although a session protocol may have only one of these message types, many protocols have both, or mix control information and data in a single message.

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. If P is embedded in Q, for instance, most packets in the session will have the format shown in Figure 6, in which the P header and data are encapsulated in Q data (the figure also shows optional footers, which are required by some protocols). In addition, the session may include control messages of Q that are independent of P and have no encapsulated P messages.

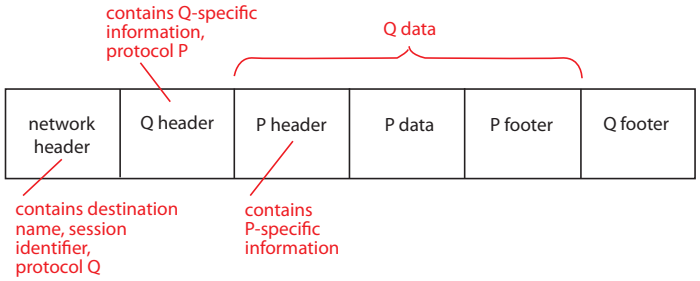

Figure 6: Packets of a network with session protocol P embedded in session protocol Q. Protocol footers are optional.

As in the figure, the first header of a packet has a format dictated by the network design, and includes the destination name and session identifier for the entire session, so that all packets of the session will be identifiable as such to the forwarders. Each network or protocol header names the type of the next header, if any, so that network members can parse and handle the packet with the appropriate protocol stack.

4.3.2 The setup phase

TLS is composed with (embedded in) TCP. If the 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 7 shows packet formats for TLS, ESP in transport mode, and ESP in tunnel mode.

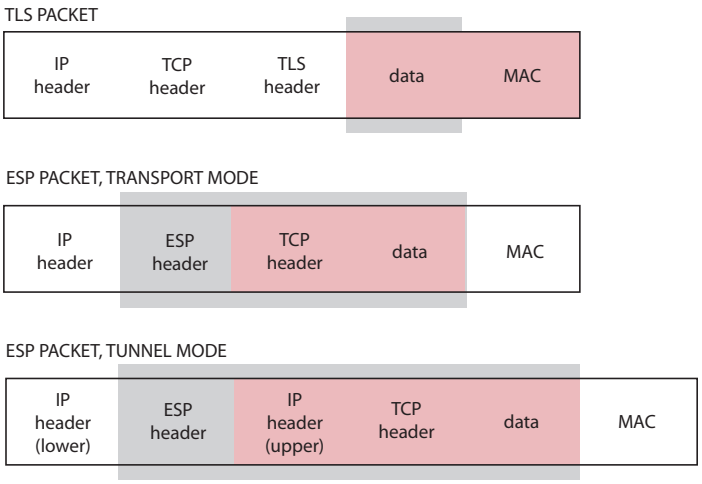

Figure 7: Packet formats for cryptographic protocols, slightly simplified. The Message Authentication Code (MAC) is a footer that assists in message authentication. Pink parts of a packet are encrypted, while gray parts are authenticated.

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 (as in §3.2). 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 (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 a session path chosen by the source of the packet. These issues will be discussed in much more detail in §6.

In a TCP/TLS session, the client and server first have a TCP (control) handshake, after which they begin a TLS (control) handshake. The TLS handshake performs three tasks: (i) endpoint authentication (§4.2.1), (ii) negotiation of a "cipher suite," and (iii) key exchange (§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. 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. Newer cipher suites are more secure, particularly with respect to key exchange that guarantees forward secrecy—learning a current key should not give an attacker clues about past keys.

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 choice of cipher suite and actual keys. Use of IKE to set up an ESP session is not always necessary because security associations can also be introduced into ESP endpoints by configuration.

4.3.3 Data integrity and confidentiality

In both TLS and ESP, data and some headers are encrypted with a shared key by the sender, and decrypted using the same key by the receiver. In both protocols, 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 (§4.2.1) to $d + k$. The MAC $H(d + k)$ is then appended to the data in the packet. As with encryption keys, both TLS and ESP 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 changes or inserts 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 7.

One might think that this problem would be solved for TLS 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. TLS packets cannot be received out-of-order because of the guarantees provided by TCP, on which it depends.

Authentication in ESP is concerned with packet replay, to the exclusion of deletion or re-ordering, because replay is the most serious security vulnerability.

ESP headers contain explicit packet sequence numbers, which are included in the data on which the MAC is computed. ESP does not have TCP to depend on, so packets could arrive out-of-order, and the receiver cannot predict the exact sequence number of the next packet. Instead, ESP checks only for received packets with sequence numbers that have already been received, and deletes them. This is sufficient to defend against replay attacks, because an attacker cannot change the sequence number of a packet it replays.

4.3.4 Uses of TLS and ESP

Properties of the protocols are summarized in Figure 8.

	ENDPOINT AUTHENTICATION using public/private keys and certificates	KEY EXCHANGE	DATA ENCRYPTION using a different symmetric key in each direction	MESSAGE AUTHENTICATION using a different authentication key in each direction
TLS	optional for initiator, mandatory for acceptor	performed by TLS handshake	IP, TCP, and TLS headers unencrypted	defends against all tampering, including replay, deletion, and re-ordering
ESP tunnel mode	optional for both endpoints	performed using separate IKE protocol, or unnecessary because pre-configured	IP and ESP headers unencrypted, encapsulated IP packet encrypted	defends against replay attacks, not deletion or re-ordering
ESP transport mode			IP and ESP headers unencrypted, TCP header encrypted	

Figure 8: Summary of protocol properties.

Use of TLS for Web traffic has been growing steadily, and now exceeds the amount using TCP. TLS is also widely used by other application protocols. ESP is most commonly used to make "virtual private networks" (see §6).

Not surprisingly, developers building applications on UDP are also interested in endpoint security. For UDP transmission, there is a security protocol called DTLS (Datagram Transport Layer Security) that is as similar as possible to TLS. DTLS introduces the notion that a sequence of UDP packets go together in a session, which is not present in plain UDP. It should be clear from the previous sections that, because DTLS is not embedded in TCP, its designers had to solve two problems: (i) the TLS handshake assumes reliable delivery of the handshake messages, and (ii) DTLS message authentication cannot rely on the property that packets are delivered reliably, in order, and duplicate-free, so that packet sequence numbers can be computed independently at each endpoint. DTLS solves the first problem by incorporating packet-loss detection and retransmission into the DTLS handshake. DTLS solves the second problem by using explicit sequence numbers, exactly as ESP does.

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 that are relatively independent of other security patterns. In considering architectural interactions, we will be looking at multiple composed networks as well as protocols within a single network.

4.4.1 Performance

Data encryption and message authentication increase required bandwidth and computational resources slightly, but no one seems to regard these as problems.

The 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. For example, the TLS 1.2 handshake consumes two round-trip times (RTTs), added to the one for the TCP handshake. Even with short RTTs, a small fraction of TLS 1.2 setups take 300 ms or more [39], due to increased computation time.

TLS handshake overhead is enough to make Web servers more vulnerable to denial-of-service attacks, as attackers can create a surge of new TLS session requests. Attempts to optimize handshakes by caching and sharing secrets among sessions have created new security vulnerabilities in TLS 1.2 [46]. In the recently approved standard for TLS 1.3, typical TCP/TLS setup times are reduced from three RTTs to two, although this may not affect the portion of latency due to computation time.

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 brief application messages can share the same TLS session. Even so, group events (such as initialization of a fleet of vehicles) can easily create spikes in the load on centralized servers [22].

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.

Architecturally, there are two ways to implement the optimization of sharing a secure channel. If TLS is being used, the application protocol could be embedded in TLS; application headers and data would be the data portion of TLS

packets, as shown in Figure 7. Alternatively, an application network could be layered on IP networks, as shown in Figure 9. The Session Initiation Protocol (SIP) is used for control of multimedia applications. The layering alternative is more flexible, because the span of each TLS session need not be the same as the span of the application session. This is essential for SIP, because an end-to-end SIP session always goes through a middlebox for each user endpoint. The two optimization alternatives correspond to ESP transport mode and tunnel mode, respectively.

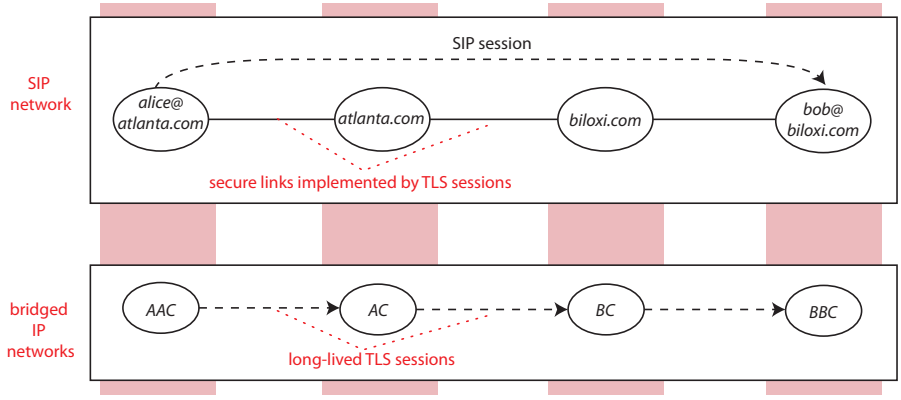

Figure 9: A SIP application network layered on bridged IP networks.

4.4.2 Session protocols

The most significant issue for composition of cryptographic protocols is their relationship to TCP, because TCP does so many things: congestion control, reliability, and packet ordering. We have seen that TLS depends on being embedded in TCP, while DTLS and ESP do not. This should not be a problem, unless real or perceived implementation constraints cause designers to make bad choices. For example, some network architectures use TCP as a session protocol in an overlay network with secure links implemented by TLS. (In comparison, in Figure 9, the overlay network uses SIP as the session protocol.) Because of the dependence of TLS on TCP, this design is layering one instance of TCP over another instance of TCP! This can cause the problem of "TCP meltdown" [23], as follows.

TCP provides reliability by detecting lost packets by means of a timer, and requesting retransmission of a packet when it does not arrive in time. For each session, TCP sets the timeout interval independently and adaptively. It can happen that the timeout interval on the upper-level instance of TCP becomes shorter than the timeout interval on the lower-level instance. In this case the lower-level session is experiencing reduced throughput, because it is waiting a

longer time for each packet. At the same time, the upper-level session is having frequent timeouts, making frequent requests for retransmission, and therefore demanding increased throughput. This mismatch drastically degrades the end-to-end throughput.

The design of the IP protocol suite is accidentally inflexible with respect to ESP. Because all networks need session identification, session identifiers should be standardized in network headers. Yet IP groups session identifiers with protocol headers, and they are not standardized across all protocols.

Specifically, the first 32 bits of a TCP header consists of two port numbers, which distinguish the session's packets from all other TCP packets with the same source and destination names, and which can easily be matched with the port numbers of reverse packets of the same session. In ESP, on the other hand, the first 32 bits of the protocol header are a pointer to a security association (see §4.3.2), which is completely different in the forward and reverse directions, and cannot be used to associate them. The consequence is that a stateful firewall or network-address translator (NAT) at the edge of a private network, configured to only allow two-way sessions initiated in the outgoing direction, will not allow ESP sessions.

In this case protocol composition enables a workaround to the problem. ESP, whether in tunnel or transport mode, can be embedded in UDP with well-known port 4500. (A well-known port for UDP/ESP composition is necessary because UDP headers have no place for the "next header," as IP and ESP headers do.) In this way a two-way sequence of UDP packets forms an identifiable session, and a stateful firewall or NAT does not see the ESP headers at all.

4.4.3 Mobility

In its strongest sense, mobility enables a session to persist even though the network attachment of a device at its endpoint is changing. This usually means that, at some level of the network hierarchy formed by layering, 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 local cellular network to another, its IP address (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 address. 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 Web site of origin delegates its identity to a content-delivery server by sharing its certificate and keys.

On the other hand, thinking about network mobility brings up a possibility we might call "reverse mobility"—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 (§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 (§2.2.2).

Unsurprisingly, some control protocols incorporate cryptography. For instance, Border Gateway Protocol Security is a security extension to BGP that provides cryptographic verification of messages 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 TLS or ESP. An endpoint may 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 protocol must have the same nonce or random value, so that no attacker without access to the previous packets of the session can send messages purporting to be part of the session. Without the nonce, an attacker could do something to trigger a query, then send a spurious answer to the query.

5 Packet filtering and resource replication

For *packet filtering*, the network infrastructure ensures that some or all packets are forwarded through infrastructure members that perform filtering. A *filter* looks for packets that satisfy its *filtering criteria.* On finding such packets, the filter takes some action as an exception to or as an addition to merely forwarding them. Packet filtering is performed by network infrastructure to defend against the security threats of flooding attacks, subversion attacks, and policy violations.

Packet filtering expends a lot of network resources, so the detailed design of a packet-filtering mechanism is resource-sensitive. For flooding attacks against servers, the goal is to use filtering resources to reduce the load on server resources. Because of this resource relationship, the design space of packet filtering extends to designs that use fewer filtering resources, but instead allocate more

resources to the servers. With more server capacity, a flooding attack does less damage. The network is usually involved, even in server-centric defenses, because the network can provide the service of distributing the load across servers. This is why we consider resource replication to fit into the same general pattern as packet filtering.

As in §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 all at the same level of the layering hierarchy, for example the bridged IP networks of the Internet. First we cover packet filtering as it is most commonly practiced today (§5.1). The next subsection (§5.2) is mostly concerned with filtering techniques that are not in widespread use today. Why are they interesting? Because the reasons they are uncommon today have to do with the current moment in the flow of technological evolution, causing certain trade-offs to be evaluated in certain ways. Technological evolution is not finished, and plausible future changes might make these filtering techniques attractive. Then, after a discussion of resource replication (§5.3), we return to the compositional view, considering how packet filtering interacts with other network mechanisms, and where it should be placed in a compositional network architecture.

5.1 Ordinary packet filtering

5.1.1 Common packet filters

The oldest filters are *firewalls*, dedicated machines positioned at or near the edges of an IP network. Their filtering criteria are Boolean combinations of simple predicates such as *destinationPort = 80* on the values of IP and IP-session-protocol header fields. Their function is to drop disallowed packets. For example, suppose that a firewall is intended to allow only outgoing Web accesses, which of course require outgoing DNS queries. The direction of a packet (inbound or outbound) can be determined from its source and destination addresses 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.

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 addresses and ports identify them as belong to an ongoing TCP session." Stateful firewalls are often combined with NATs (§4.4.2), because NATs sit at the edges of networks and already maintain tables of ongoing sessions. 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."

Figure 10 is a table summarizing some characteristics of four common types of packet filter in IP networks. We want to stress that the classification is at least as much historical and marketing-oriented as it is technical. The table should not be read as prescriptive, but merely as an illustration of various options.

	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
ACTIONS TAKEN	drop packets	drop packets	raise an alarm, divert packets for further analysis	drop packets, rate-limit packets, refuse requests, record packets
REQUIRE SESSION AFFINITY	no	yes if stateful	yes	yes
CAPACITY	dedicated, high-capacity machine	dedicated, high-capacity machine	virtualized for dynamic scale-out	virtualized for dynamic scale-out

Figure 10: Examples of common packet filters in IP networks.

"Capacity" in the table refers to filtering resources. How does a network ensure that its packet filters themselves do not become traffic bottlenecks during flooding attacks? Firewalls run on large machines, with capacity sufficient to handle all their network's traffic, even during a flooding attack.

Sometimes "routers" (the forwarders in an IP network) are also used to filter packets because they are located on paths inside a network. Routers are expected to work even faster than firewalls, so they do not perform stateful filtering. Their collections of filtering rules are called "access control lists."

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." The difference 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.

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. When filtering is being used to defend against policy violations, sometimes the filter records packets for the purposes of investigation and legal evidence.

5.1.2 Filtering criteria

Filtering criteria choose the packets on which a filter takes action. Beyond the simple header inspection performed by firewalls and routers, intrusion detection and prevention systems usually look at packet data. The problem of finding effective filtering criteria is different for the different purposes of filtering.

If the purpose of filtering is to prevent subversion attacks or find policy violations, then the filtering criteria must describe some characteristics of the subversion or violation. Firewall predicates describe policy violations. "Signature-based" filters such as spam filters, virus scanners, and parental filters look for keywords, sometimes keywords in specific positions, and other known attack patterns. The 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. Even so, they cannot detect new attacks.

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 [21].

One advantage enjoyed by filters for preventing subversion attacks or finding policy violations is that attackers cannot usually hide by spoofing. If the attack requires communication in both directions, then the attacker's source name must usually be correct. It takes a very sophisticated attack, such as BGP hijacking

(§2.2.2), to get packets delivered to a spoofed source address.

If the purpose of filtering in IP networks is to prevent flooding attacks, it must contend with the fact that packet source names can be false (the same is true of other networks that allow spoofing, such as email application networks). On the other hand, at least having a false source name is a straightforward packet-filtering criterion, if the filter can detect it. "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 the user machines with members in it, 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 forwarding cannot be used in the high-speed core of the Internet, because routes there are not necessarily symmetric.

The individual packets of a flooding attack look benign, so attacks are detected by means of statistical algorithms. These algorithms look for anomalies, *i.e.,* variations from normal patterns of bandwidth use, protocol use, and other traffic attributes. Accurate detection of flooding attacks is difficult for many reasons; in addition to the many ways that attackers can hide (§2.2.1), there may be unavoidable congestion due to failures, or a legitimate flash crowd [36]. Detecting too many anomalies ("false positives") causes collateral damage, as many legitimate packets may be filtered out.

In general, the quality of filtering criteria is a limiting factor in the use of filtering to handle IP flooding attacks. Consequently, some of the research on flooding attacks aims to make filtering criteria precise by recognizing certain packets as desirable and rejecting all other packets. We'll call this approach "positive filtering" because the default action on a packet is to drop it, and matching a filtering criterion allows the packet to be delivered. In addition to precision, positive filtering claims the advantage of preventing flooding attacks, rather than reacting to them well after they have begun.

The limitation of positive filtering is that it only works in a constrained context, where desirable packets can be recognized. For instance, Secure Overlay Services [28] is a positive-filtering proposal for public emergencies, in which all normal traffic is suspended and protected servers should be reached by emergency responders only. Another example, Ethane [10], is intended for private IP networks with software-defined control. An Ethane controller is a central network member with very complete knowledge of its network, especially the user members. For each user member the controller knows the IP and MAC addresses, the forwarder port to which the member is directly attached, and the user of its machine. It also has policies governing which user members can reach which user members, the session protocols they can employ, and the middleboxes the sessions must pass through. When an Ethane forwarder receives the first packet of a session, it sends the packet to the controller, which uses its knowledge and policies to choose to either allow or disallow the session. If the

session is allowed, the controller installs a tuple for it in the forwarding table of every forwarder in the path of the session.

In conclusion, the hardest attack target to protect is a public server, because its job is to serve millions of previously unknown clients from all over the world, and the business success of its enterprise depends on continuous high-quality service.

A recent flooding attack on a number of DNS servers [15], 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 (1200 Gbps). To provide some intuition about the resources needed to handle such attacks, the table shows some typical capacities for servers and various kinds of packet filters. Of course these numbers are subject to frequent change, as 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 conventional hardware.

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

Figure 11: Data-processing capacities of common packet filters and servers.

5.2 Path-based packet filtering

Attack signature is our general term for a predicate that distinguishes packets belonging to one attack, as best as network tools and operators are able to diagnose it. Path-based packet filtering augments attack signatures based on the contents of packets, as discussed in the previous section, with predicates based on the path along which packets traveled.

There are two reasons for introducing path-based filtering, especially for flooding attacks. The first reason is that path information can improve the precision of attack signatures, 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 attack signature based on packet contents. If we know that most paths to the server are delivering a trickle of these packets, and one path's capacity is dominated by them, there is a good chance that only the packets on the dominated path are attack packets.

The second reason for path-based filtering is that it may be possible to filter out attack packets closer to their sources, which reduces the damage they do.

This section will discuss the trade-offs, why path-based filtering is not much used today, and why it might be used more in the future.

5.2.1 Tracing attacks back to their sources

In IP networks, because of spoofing, the source name in a packet is not a reliable indicator of where it came from. The purpose of a traceback mechanism is to determine the path along which an attack packet travels. In other words, traceback associates path meta-data with packets. Note that the Accountable Internet proposal (§4.2.1 recommends authenticated source names so that, among other reasons, traceback is not needed.

From the viewpoint of a victim of a flooding attack, the network is a directed acyclic graph with many possible packet sources and a single packet sink, which is the victim. Often the graph is a tree, with the victim at the root. The interior of the graph consists of forwarders and middleboxes, connected by links carrying traffic toward the victim. Figure 12 shows such a graph for attack victim T. In the figure, member names also stand for names of machines.

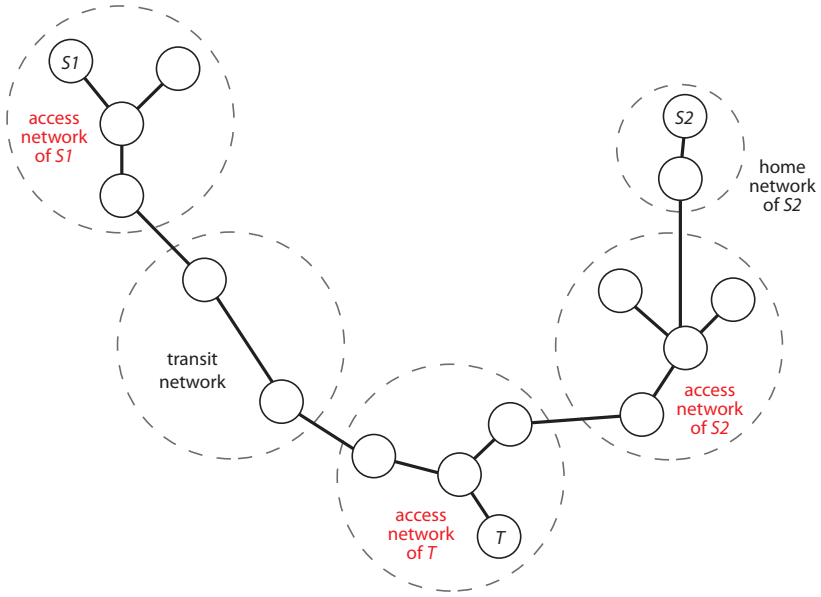

Figure 12: Paths from packet sources $S1$ and $S2$ to an attack victim T.

Figure 12 illustrates some distinctions that are relevant to path-based filtering. The *access network* of a machine is the first network in all its outgoing paths whose AA is different from the owner of the machine (assuming just for simplicity of discussion 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 has a member of its access network, as T and $S1$ do. Sometimes, however, the first network of a machine (for example $S2$) is a *home network* whose AA is the same as the owner of the machine. In this case the machine has separate home and access networks.

The simplest traceback mechanism is the Internet Control Message Protocol Traceback message. The idea is that each forwarder samples the packets it is forwarding, with a very low sampling rate. When a packet is chosen, the forwarder encloses the whole packet, along with the names of itself, the preceding forwarder, and the succeeding one, in a Traceback message, and sends it to the packet destination. The idea is that the victim or a nearby helper will reconstruct whole paths and maintain a running view of where its packets are coming from. In addition to cumulative overhead, the chief disadvantage of Traceback messages is that attackers could forge them. To prevent this, the sources of Traceback messages must be authenticated with public-key cryptography, introducing significant additional overhead.

In other traceback mechanisms, forwarders mark the packets themselves with path information as they are forwarded toward the victim. The markings allow the victim or a helper near the victim to reconstruct the path along which packets of an attack traveled.

The design of these traceback mechanisms entails many trade-offs. In the remainder of this section we use representative proposals to illustrate the issues and indicate some of the trade-offs, without declaring any particular winners.

Internet measurements from 2000 indicate that the average-length path has 15 forwarders, and 40 is a practical maximum [49]. It would take a large amount of space in packets to represent full paths. One design allocates space in packets to record only a single forwarder (32 bits for an IP name), and uses probabilistic marking, in which each forwarder marks a packet with probability p, say 0.04 [44]. Because forwarders late in the path overwrite the marks of earlier forwarders when they mark, the forwarders in a path can be ordered by the frequency of their marks. The disadvantages of simple probabilistic marking are:

- A path cannot be reconstructed until hundreds or thousands of attack packets have been received.
- If an attack packet traverses its entire path without being marked, the contents of its mark field is whatever the attacker put in it. The attacker can put deceptive information in this field.
- The attack signature may very well include attack packets from more than one source, because of botnets and coordinated attacks. In this case simple probabilistic marking will fail, because mark frequency is not enough information to reconstruct multiple paths.

The final IP Traceback design [44] deals with the problem of multiple attack paths in simple probabilistic marking by encoding edges in the tree rather than

nodes. The resulting doubling of the space needed for marks is dealt with by very aggressive compression techniques, primarily making each mark field contain only a fragment of the full-size mark. This reduces the mark field to 16 bits, but has the effect that many more packets must be received before the path can be reconstructed. Even so, simulations show that a path of length 15 can almost always be reconstructed after the victim or its representative has received 2500 attack packets. The disadvantages of IP Traceback include attacker deception, as above. More importantly, the reconstructed tree of multiple attack paths becomes the solution to a large combinatorial puzzle. The puzzle can be solved only with data from many received packets, and only using a large amount of memory in the component solving it.

The Pi design [49] has every forwarder (at least, within a specified path segment) mark the packet. Because the mark encodes the entire path (or segment), the victim's helper need only receive one marked packet to have all the information available about the path. Deterministic marking is combined with very aggressive compression of the mark field, again down to 16 bits. Primarily, compression in this proposal means that more than one path can result in the same mark. The disadvantages of Pi are:

- Marks do not have enough information to reconstruct paths, only to distinguish equivalence classes of paths (all the possible paths that happen to map to the same mark). So marks are not helpful in locating or distributing packet filters—all the filtering must be done near the victim.
- Marking and filtering require choosing three parameters, each difficult to choose in general and having interactions with many other factors, including the ability of attackers to inject deceptive information.

In Active Internet Traffic Filtering [3], every packet is marked with full node names, but usually only by the egress forwarder at the edge of the source's access network, and the ingress forwarder at the target's access network. If a source/destination pair seems to distinguish an attack, a controller in the target's network will request both the target-network forwarder and the source-network forwarder to drop such packets (see below). The source-network forwarder includes a nonce in the mark, and the controller copies it into the request, which is how request messages are secured (as in §4.4.4).

5.2.2 Filtering upstream

In the Internet, at any given time, there is a relatively small number of targets for active flooding attacks. To defend a target against these attacks, packet filters can conceivably be placed in the graph of paths downstream, near the target, or upstream, near packet sources. There are three main advantages to placing packet filters upstream:

- 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 packet 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.

A third option, filtering in the topological core of the network, is never used because the core is a region of high-speed links and high-speed routers handling large numbers of packets. The required speed of filtering, and the potentially large number of filtering rules to be checked, makes this option infeasible.

Pushback [36] is a simple scheme for reducing overall congestion by pushing filtering upstream. At a forwarder, congestion on an outgoing link is diagnosed when there is frequent packet loss (packets must be dropped because there is no room for them in the link's output queue). If a particular aggregate of packets is responsible for a significant portion of the link's traffic, then a predicate describing the aggregate becomes an attack signature. The forwarder sends upstream, on all its input links carrying packets in the aggregate, a request to rate-limit these packets. Upstream forwarders can also request pushback recursively, so pushback incorporates its own traceback mechanism. By rate-limiting only specific aggregates along specific paths, pushback aims to do just enough to protect other traffic, while limiting collateral damage.

The Active Internet Traffic Filtering proposal [3] employs upstream filtering because it is particularly concerned with the botnet case, and with using the many forwarders in the access networks of attackers to help filter. There are several ways in which its basic idea (above) must be augmented to make it reliable and secure. First, the request and acknowledgment packets of the control protocol itself could be used to flood a network, so they must be rate-limited. Second, there is a set of mechanisms through which forwarders are monitored to see if they are keeping their filtering promises, and through which filtering can be delegated to other forwarders along an attack path.

5.2.3 Capabilities

In the long struggle to defend public servers against flooding attacks, researchers have explored an alternative approach based on "capabilities" (a capability is an unforgeable record showing the rights of the bearer). The idea behind capabilities is that no source should be able to send Internet packets to a destination unless the destination has already approved the transmission.

As applied to the protection of a public server accessed through TCP, the TCP SYN is interpreted as a send request to the destination. Unless the source is already on a blacklist, the server will grant permission to send a limited number of packets in a limited period of time, and reply to the SYN with a capability attesting to the permission. The sender includes the capability in subsequent packets, and forwarders on the path enforce the capability policies. The destination can grant permission for more packets later, if they are needed and the source has been well-behaved. Packets with no capabilities, expired capabilities, or incorrect capabilities *may* be delivered, but with the very lowest priority.

An example of this approach is the Traffic Validation Architecture [50]. The architecture includes elaborate mechanisms to ensure that capabilities cannot be forged by attackers, and cannot be transferred to other attackers. It includes mechanisms to reduce the amount of space needed in packets for capabilities, which can be considerable. It also has mechanisms for reducing the amount of state in forwarders required to implement the security measures and to track packets sent and time elapsed.

The principal problem with capabilities is session requests, which cannot be controlled with capabilities and can be used on their own to create flooding attacks. The Traffic Validation Architecture handles this problem by rate-limiting request packets to 5% of the total volume. It can be shown, however, that this just turns a flood of request packets into a denial-of-capabilities attack, in which legitimate senders cannot get their requests through [4].

5.2.4 Filtering downstream

The advantages of filtering upstream are balanced by two major 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 [3]. Historically, however, cooperation between networks with different AAs has been scarce [20].

- Even if source networks are willing to cooperate with target networks, the necessary coordination is not easy. Previous sections have illustrated many forms of overhead and many security vulnerabilities introduced by the coordination, and we have not even yet mentioned the issue of backward compatibility.

The proposals for moving filtering toward the sources of attack traffic date from the early 2000s. In the 2010s cloud computing advanced so far that it altered the evaluation of trade-offs decisively. Now almost all packet filtering is performed on behalf of the access networks of attack targets. It is usually performed in clouds, with the filters running on virtual machines, so the number of filters can expand and contract with fluctuations in load. The same routing

flexibility used in clouds to allocate more virtual machines to a task can also be used to deploy sequenced filters. With sequenced filters, packets that simple filters find suspicious can be deflected to complex filters for more detailed screening.

At this moment the reader might be wondering why we went through all the detail of §5.2.1 through §5.2.3 if most of it is irrelevant today. The point is that it was made irrelevant by technology changes that altered the evaluation of tradeoffs, and technology changes in networking are not finished. Future changes could easily make old solutions interesting again. Here are three examples:

- If the Internet evolved to offer more paths with reserved bandwidth for real-time applications, then capabilities might be an excellent approach to securing the use of reserved paths.
- As individual IP networks grow in size and geographical scope, 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 AA. If so, then the administrative barriers to upstream filtering will disappear.
- Most traceback proposals require IP forwarders to perform new functions—marking and filtering packets in new ways. The logic in most IP forwarders has been and still is fixed in the factory. Now that programmable forwarders are coming into more common use, it will become much easier for network operators to experiment with traceback and other such schemes.

5.3 Resource replication

Just as cloud computing has made filtering near attack targets scalable and therefore attractive, cloud computing has also made it feasible to defend against flooding attacks by replication of user resources. When a service is under attack, it can quickly be granted more virtual machines.

It is even 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

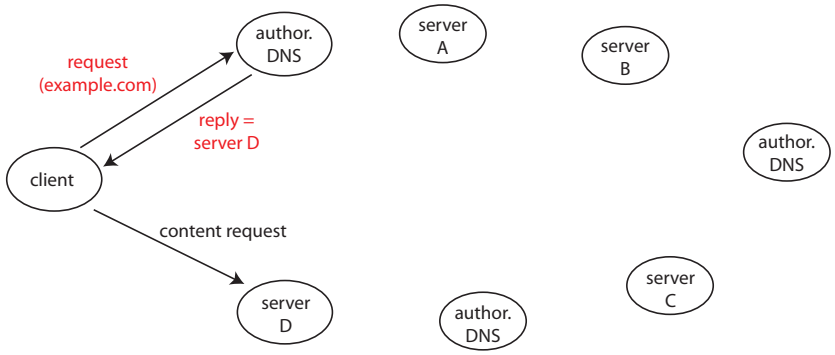

Figure 13: Resource replication in a content-delivery network. All member labels symbolize IP addresses, so the three DNS servers have the same address.

servers. So each DNS server can return to a client the IP address of the best content server for it to contact.

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, as will the expected availability of an arbitrary key.

Another way that a distributed system or service using a network can protect itself against flooding attacks is by reducing the amplification factor on which flooding attacks depend. For example, It has become common for servers to defend themselves against SYN floods (§2.2.1) with "SYN cookies." In this defense, the server returns a SYN+ACK packet with 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 not been amplified. 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.

There is another self-defense against SYN floods that is less efficient than SYN cookies, but comes with fewer side-effects (see §5.4.4). This defense uses a middlebox in the path to a Web server that stores and responds to SYN packets, but does nothing else with a SYN packet until it receives the ACK that

completes the handshake. On receiving the ACK, the middlebox forms a new session by sending the SYN to the server, and subsequently acts as a transparent forwarder between the two sessions. If the middlebox does not receive a timely ACK, then the SYN packet was part of an attack or the client has failed, so the middlebox drops it.

For another example of service self-protection, 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 [38]. 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. The same idea would work for many other services with caching.

5.4 Interactions between packet filtering and other aspects of networking

5.4.1 Routing

For a filtering tree or graph (as in Figure 12) 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 "waypointing" constraints about steering packets through filters [6, 7, 16, 27, 35].

Another issue that complicates routing through a filtering tree is the fact that many packet 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.

5.4.2 Layering

Almost always, a packet arriving at a machine is being transmitted through multiple layered networks simultaneously, for example an Ethernet LAN, an IP network, and an application network. Figures 1 and 2 combined illustrate this simple example. Amplification of a flooding attack, or damaging processing of a malware packet, can take place at any of these levels. Filtering can also take place at any of these levels.

The simple case is not necessarily the most common today. Layered between the application network and the IP network there is often a virtual private network (§6.2.2). 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 LAN, with the job of sharing cloud resources among all tenants.

In this section we discuss two interactions between packet filtering in a network (or layer of bridged networks) and layering. The first interaction concerns networks below the filtering network in the layer hierarchy, and the second interaction concerns networks above the filtering network.

Imagine that you have designed a filtering mechanism within a network, and proved that it is correct. Your proof concerns (among other things) paths in the network to a potential attack target, and shows that routing places an appropriate packet filter in every path.

Whether you remembered to state it or not, your proof 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 packet 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. It is easy to make a penetrated Ethernet inject packets into the links of networks layered on it [30]. 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.

The second issue concerns layers in the architecture above the network where filtering takes place. It is common to deploy IP-based intrusion detection systems that look into IP packets for attack signatures at the application level, for example signs of application malware at particular locations in TCP payloads. These systems are assuming there are no networks layered between the filtering network and the application network. If there are additional networks, then the filtering criteria will be useless. The ideal solution to this problem would be to filter packets separately in each network, with each filter being attuned to the protocols, vulnerabilities, and configuration of its particular network at its level

in the layer hierarchy. It should be possible to optimize network designs so that filters at multiple levels are frequently located on the same machines.

5.4.3 Cryptographic protocols

A *middlebox* is a network member—other than a forwarder—in the path of a session. Packet filters are middleboxes. Most middleboxes are infrastructure members of a network, performing a wide range of useful functions for performance optimization and interoperation as well as security.

There is a profound interaction between cryptographic protocols and packet filtering in a network. If a user session is encrypted, then middleboxes in general, and packet filters in particular, cannot read anything in the session packets beyond their headers.

First we cover the interested parties, their powers, and how their powers interact. Later we present specialized techniques for managing the interaction in cooperative cases.

Interested parties and their powers

Because all the parties interested in a user session have different powers, we can think of their interactions as a game, one instance of which is illustrated by Figure 14. 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 cooperates, the data of the session will be encrypted end-to-end.

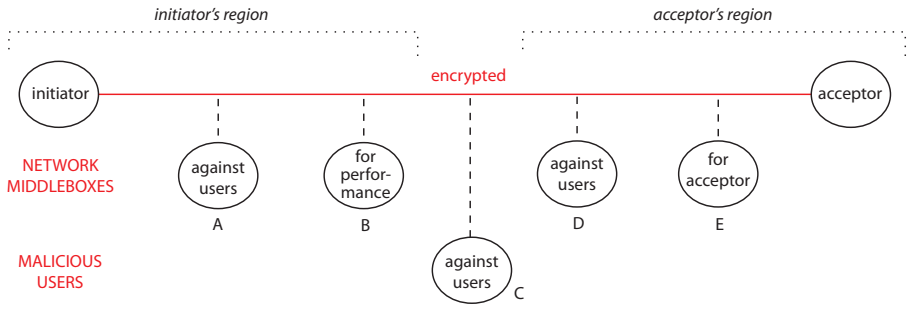

Figure 14: A game: cryptographic protocols *versus* packet filtering.

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 packet 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 packet 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 users 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 user is able to eavesdrop in the middle of the session path, but not near the endpoints.

It may be that encryption will prevent all five middleboxes from achieving their goals. Middleboxes B and E may be the easiest to help, because the goals of the middleboxes coincide with the interests of the users (see below).

For middleboxes D and E, the network could introduce another middlebox that will act as endpoint for the initiator's TLS session. A middlebox that is a session-protocol endpoint is called a *proxy*. The proxy would accept the initiator's TLS session and make a TCP session between itself and the original acceptor. The proxy would decrypt packets from the initiator and send their contents in plaintext packets to the acceptor, so they could be read by any middleboxes in the path of the TCP session. If the proxy is placed before D, then both D and E will be able to do their jobs.

In all cases middleboxes A and C will be prevented from achieving their goals, unless they can achieve them by reading headers alone.

Cooperative encryption

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.

If network middleboxes must belong exclusively to general-purpose IP networks, then another possible approach is represented by Middlebox TLS (mbTLS) [40]. In this approach all middleboxes must be proxies, and they create a session consisting of an end-to-end chain of simple TCP sessions. Along the chain from initiator to acceptor, there is first a set of middleboxes inserted on behalf of the initiator, followed by a set inserted on behalf of the acceptor (see Figure 15).

Within the end-to-end chain of TCP sessions, the initiator and acceptor first have a normal end-to-end TLS handshake for endpoint authentication and key exchange. Then each middlebox initiates a secondary TLS handshake with the next element in the direction of its sponsoring endpoint. For example, if there are two middleboxes *M1* and *M2* inserted on behalf of the initiator, *M2* initiates a secondary handshake with *M1*, and *M1* with the initiator. The secondary handshakes exchange symmetric and authentication keys for the individual simple

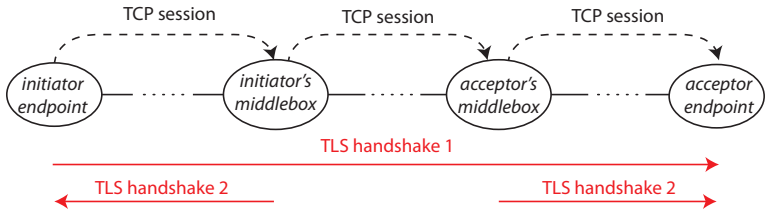

Figure 15: Control signaling to set up an mbTLS session.

sessions. They can also perform endpoint authentication of middlebox identity (in this case the responsible owner), software version and configuration, security properties of the hardware/software platform, *etc.* This makes sense because the middleboxes associated with each endpoint are working in cooperation with it, even if they have an adversarial relationship with the middleboxes of the other endpoint.

After the secondary TLS handshakes, data is transmitted. In each middlebox data is decrypted, processed as plaintext by the middlebox application code, then encrypted again for the next hop. Note that the "midpoint" simple session between initiator and acceptor middleboxes has no secondary handshake; in this simple session, the keys chosen by the primary TLS handshake are used. An earlier version, Multi-Context TLS [41], allowed the endpoints to place constraints on the read/write access of middleboxes.

The third approach is based on new results in cryptography. At one extreme, fully homomorphic encryption [18] 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.

BlindBox [47] is a proposal for allowing middleboxes to operate on encrypted data. BlindBox middleboxes can apply detection rules of the kind commonly used by virus scanners, intrusion-detection systems, and parental filters. These rules often identify keywords and other exact strings at particular positions in the data, which the middleboxes can search for. The scheme also allows a middlebox that has found a keyword or other suspicious string, as probable cause of a security violation, to decrypt the entire packet.

BlindBox is implemented as an extension of TLS. In addition to the basic TLS handshake, endpoints must generate extra keys. The data must be sent end-to-end twice (redundantly), once in the ordinary TLS form and once in a reformatted and re-encrypted form suitable for the Blindbox algorithms.

The biggest overhead incurred by Blindbox is due to rule preparation, because the middlebox must have the rules themselves encrypted with a session-dependent key. The endpoints must not know the rules (this would make them

easier to evade) and the middlebox must not know the key (otherwise the guarantee of confidentiality would be lost). So who can encrypt the rules? For every keyword in every rule, both endpoints must generate and transmit to the middlebox a special encryption function that incorporates yet obfuscates the session-dependent key. The middlebox must first check the two for agreement (in case one of the endpoints is insecure) and then apply the encryption function to the rule. This results in very high performance overhead, which means that Blindbox is currently practical only for long-lived sessions or small rule sets.

Although both Middlebox TLS and Blindbox are promising efforts, it seems clear that their complexity and non-uniform communication among participants are weaknesses. Complexity itself is a security vulnerability, because it provides a larger "attack surface" for adversaries to probe.

5.4.4 Session protocols

There is a general problem affecting all protocols of the IP protocol suite: When a new network feature requires additional information in packets, where can the information be put? Inventors do not wish to increase packet size, and want their proposals to have backward compatibility. So they generally choose to fit their information into some "unused" field in current header formats. For example, both IP Traceback [44] and Pi [49] squeeze traceback information into the 16-bit identification field in the IP packet header. The original use of this field is to group fragments of a fragmented packet so they can be re-assembled at their destination. It is declared "unused" on the grounds that fragmentation of IP packets is now rare.

The irony of such proposals is that so many new features and protocol variations use the same "unused" fields. At the same time, inventors of other new functions have no compunction in deleting this "unused" information from packets when it is convenient. For example, a server using SYN cookies (§5.3) effectively drops all optional information in TCP SYN packets, which means that any network feature relying on extensions to TCP is disabled. Note that many servers using SYN cookies are Web servers, and many new features relying on extensions to TCP (such as Multipath TCP [42], just to name one example) have improved Web access as a major use case.

For those who harbor hope that the Internet will be better in the future, even if the improvements are not backward-compatible, we believe that the right solution to such problems is a generalized mechanism for composition of session protocols. This mechanism would build on the structures of Figure 6. If all session protocols could be composed freely, then all new features requiring space in packets could be introduced—without fear of interference—as new, composable session protocols. For instance, just as TLS is composed with (embedded in) TCP, TCP could be composed with (embedded in) Multipath TCP. TCP would control transmission of byte streams along individual paths, while Multipath TCP would coordinate the multiple byte streams. As for increases in packet size, there are already well-established techniques for handling packets that exceed the maximum packet size of a network.

6 Compound sessions and overlays for security

Like cryptographic protocols, compound session and overlays are mechanisms employed by users to defend themselves against spying and tampering attacks. Cryptography alone is not sufficient because it does not conceal packet headers (§2.2.4).

The focus of this section is on middleboxes. So far the middleboxes we have seen are infrastructure members of a network, and are inserted into session paths by means of routing. Most importantly, packet filters are middleboxes.

It is also possible for a user member initiating a session to 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 Figure 16. The middlebox must learn the initiator's intended destination, for example by getting it from the payload 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. A middlebox that behaves in this way is called a *joinbox*. Each joinbox 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* associated by joinboxes in this way.

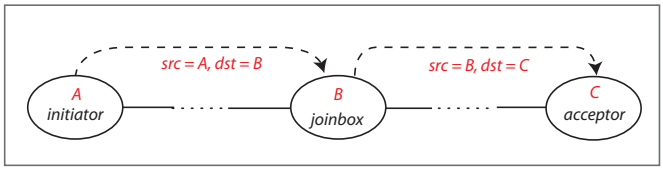

Figure 16: A compound session with two simple sessions.

A compound session can have more than one joinbox (as an example of how to do this, the session-initiation packet can contain a list of joinboxes to visit, ending with the final destination). Because of the names in forward packet headers, return packets naturally pass through the same joinboxes in reverse order, and have their headers re-translated in reverse order.

The behavior of a joinbox is not limited to the behavior described above. In particular, often a joinbox is also a proxy (session-protocol endpoint) as introduced in §5.4.3. If a joinbox is also a proxy, it can get the name of the next joinbox or acceptor by using the session protocol to engage in a dialogue with the initiator. If a joinbox is also a proxy, it can perform other kinds of computation on packets, even translate from one session protocol to another, so packets in its two associated simple sessions need not correspond one-to-one.

The 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 (which is why they are complementary to cryptography). In Figure 16, an observer between A and B cannot observe the true acceptor of the compound session, at least from packet headers alone, and an observer between B and C cannot observe the true initiator of the compound session.

Spying and tampering attacks can be launched by other user members of a network (authorized or unauthorized), and, most importantly, by infrastructure members of a network performing packet filtering. Thus the entire topic of this section can be seen as an interaction between two patterns, namely compound sessions and packet filtering.

We can think of this interaction as a game. Within a network in which there is packet filtering, the user tries to evade filtering by creating a compound session such that the simple session routed through a filter has a header obscuring the relevant information. For example, if there is a destination-sensitive filter between A and B in Figure 16, it will not see the true destination of the compound session. The network tries to prevent this, either by detecting joinboxes and blocking all traffic to them, or by placing filters in all simple sessions. The game becomes much more interesting when a compound session traverses bridged networks, because some transit networks may be friends of the initiating user, while others are friends of the initiator's access network. Considering all the transit networks, and all their user members acting as joinboxes, the game has many interested parties—some being adversaries of each other, and some cooperating with each other.

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.

6.1 Compound sessions

6.1.1 Joinboxes in access networks

Perhaps the oldest example of a joinbox for evading packet 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 the 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 (and the mapping may not even be static).

An application gateway for the application, for instance Telnet, solves this problem, as shown in Figure 17. To use it, a user initiates a Telnet session to the application gateway inside the enterprise network. The gateway is a Telnet proxy, as well as a joinbox. 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.

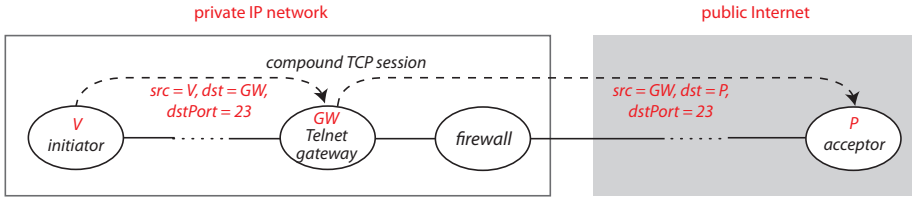

Figure 17: A Telnet application gateway inside the access network of V.

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

In a similarly cooperative situation, the AA of a private network might provide a proxy (with a public name) that session 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.

6.1.2 Joinboxes in transit networks

A user can evade filtering in his access network more easily by connecting to a friendly joinbox in another network. This will be illustrated by the use of a joinbox/proxy to reach a Web server. This kind of proxy is sometimes called a "virtual private network."[1]

In Figure 18 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 proxied 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).

[1] Calling a proxy a network is a misnomer. See §6.2.2 for the real thing.

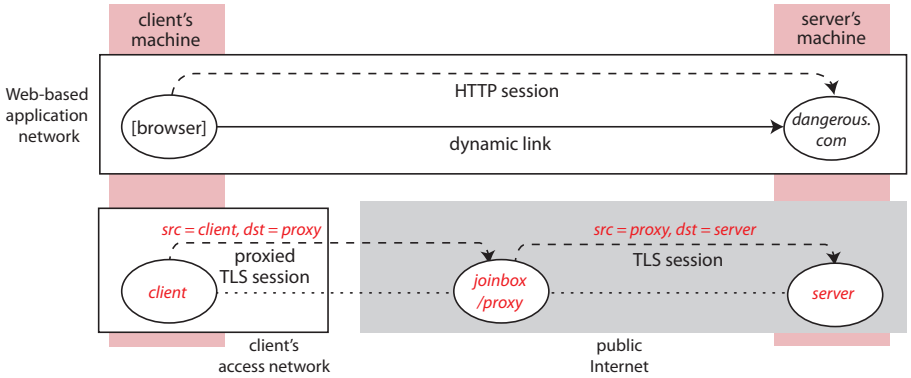

Figure 18: A proxied TLS session protects the client's privacy in his access network, and provides anonymity at the Web server.

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 addressed to external proxies. These disadvantages are addressed in subsequent sections.

The proxy in Figure 18 is specialized to handle TLS sessions. There can be proxies for other session protocols as well. For example, a "recursive" DNS resolver is just a proxy for the simple request/response session protocol used for DNS lookups. ODNS [45] is a proposal for improving the privacy of users doing lookups by introducing a proxy between local DNS resolvers and authoritative resolvers.

To introduce the proxy, as shown in Figure 19, the user appends—to the domain name it intends to send in its request—the extra high-level domain name `.odns`. This will cause a local DNS resolver to send the request to an ODNS proxy. To conceal the true desired domain D from the local DNS resolver, the user generates a symmetric key k, encrypts D with k, and encrypts k with the public key K of ODNS proxies. A concatenation of these two values is the domain name it sends in its request. The ODNS proxy decrypts with its private key to get k, and then decrypts with k to get D. After getting a response from an authoritative server for D, with an IP name N for D, it encrypts both D and N with k, and sends them in a response to the local DNS resolver. As a

result of this design, the local DNS resolver will not know what domain name is being looked up, and the ODNS resolver will not know the identity of the user.

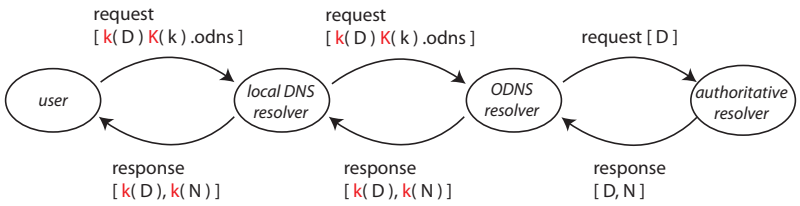

Figure 19: Oblivious DNS lookup for user privacy.

6.1.3 Deflection

The problem that a censoring access network can block packets to known proxies has been addressed by several similar proposals [24, 26, 48]. They all use joinbox/proxies, but in a way that still works despite the blocking.

A typical compound session in these proposals is shown in Figure 20. 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 18.

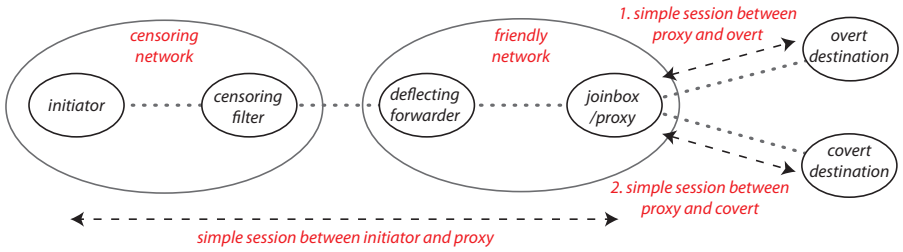

Figure 20: A deflected compound session between an initiator and a covert destination. In the simple session on the left, names in the IP header are those of the initiator and overt destination; packets from the initiator are deflected to the proxy as an exception to normal forwarding.

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 [24], the user registers with the friendly network; while the registration is active, all sessions initiated by the user are deflected. In decoy routing [26], this is done on a session-by-session basis. Decoy routing assumes that the initiator and proxy share a secret key, from which a set of nonces can be generated. As a hidden deflection signal for a session, the initiator places a generated nonce in a pseudo-random field of the first TLS message.

The TLS-based session protocol between the initiator and the proxy is complex. 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 endpoints (and its friends). The AA 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. In this game BGP inter-network routing helps the endpoints more than the censoring network, because it gives the censoring network only a single route to each destination.

The rules of this game may change in the future: new versions of the Internet may give path-selection control to user members of networks, and there may be explicit associations among sets of cooperating networks, both of which are recommended by the SCION project [5]. 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.

6.2 Overlays

An overlay is a virtual network layered on top of an underlay network (§3.2). We will first summarize the differences between overlays and compound sessions,

then show their use in three security designs.

6.2.1 Overlays *versus* compound sessions

Figure 21 shows a prototypical overlay session whose links are implemented by sessions in one or more bridged underlay networks. All four machines belong to user members of their underlay networks. From the viewpoint of the underlay networks, this looks very similiar 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 joinboxes), 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 can be used 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 (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 §4.4.1).

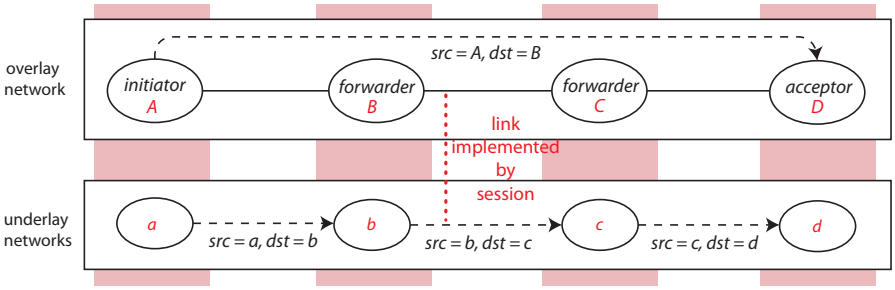

Figure 21: A prototypical overlay session.

6.2.2 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 22.

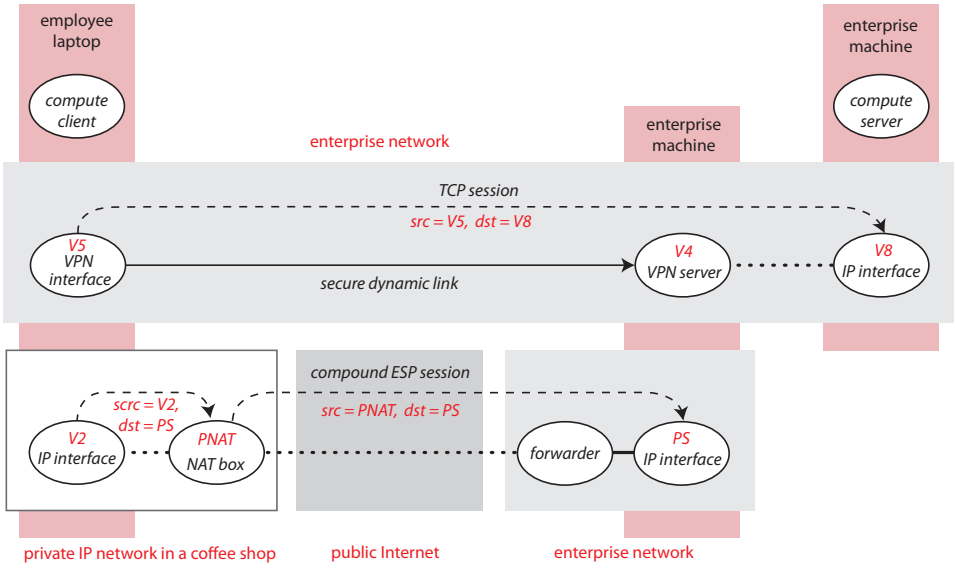

Figure 22: An enterprise network using VPN technology. A secure dynamic link in the enterprise network is implemented by an ESP session in tunnel mode.

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 (§4.3.2, §4.4.2) to public IP address *PS*, which is part of its configuration. 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 NAT box (a joinbox) 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, packet 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.

6.2.3 Overlays for positive filtering

In §5.1.2 we introduced positive filtering (filtering in which the default action on a packet is to drop it) as an interesting technique with limited applicability, because of the difficulty of finding precise filtering criteria. Positive filtering is similar in spirit to capabilities, introduced in §5.2.3, but it turned out that denial of capability is as much of a problem as denial of service. Several researchers have explored whether the properties of overlays can be exploited to make a success of positive filtering.

In both Mayday [1] and Secure Overlay Services (SOS) [28], the initiator of a session is authenticated, and session packets are not transmitted unless authentication succeeds. To understand these proposals, it is best to imagine the perspective of an access network AA trying to protect a Web server within the network from flooding attacks. The obvious problem with authentication at the edge of the access network is that the authenticators are a limited resource, easily overcome by denial-of-capability attacks. Mayday and SOS use overlays to extend the geographical span of the access network, enlisting allies all over the public Internet to perform authentication. The authenticators are the overlay ingress nodes in Figure 23. The idea is that there can be enough authenticators, dispersed widely enough, to resist flooding and denial-of-capability attacks. Even though the authenticators can be way upstream of the protected target, this is different from upstream filtering (§5.2.2) because the authenticators are user members of the IP networks in which they reside, not infrastructure members.

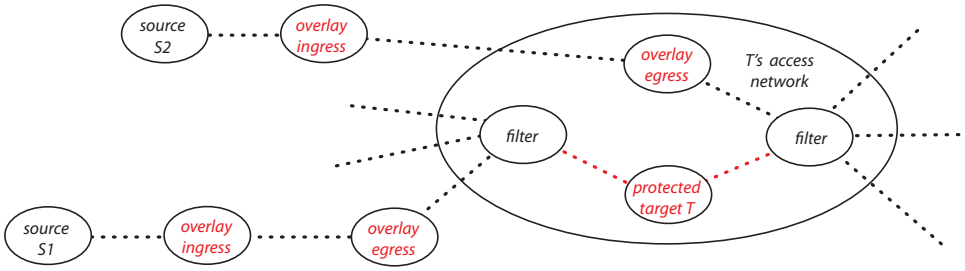

Figure 23: A network graph of Internet members involved in overlay-based positive filtering. The members named in red are on overlay machines, *i.e.,* their machines also have interfaces to the overlay network. The paths in red are the *only Internet paths* to the protected target.

The general idea of Mayday and SOS is that packets of authenticated sessions travel to the protected target through the overlay as well as the Internet. Packet filters near the protected target can distinguish overlay packets, and discard all

other incoming packets. Details are given below.

Most importantly, these proposals make use of another overlay property in addition to flexible span: because an overlay is a network, it has its own routing. Overlay routing is used to vary and hide the paths of packets between ingress members and the target. This keeps attackers from flooding the paths to the target rather than the target itself. For instance, SOS uses a complex routing scheme, with long paths computed from distributed hash tables, for path secrecy. Other implications of overlay routing will be discussed further below.

In designing an overlay network for positive filtering, there are three important choices to be made. SOS makes specific choices, while the Mayday paper points out that there are other choices, and evaluates some combinations of them. We now explain the three choices.

Source authentication

This choice concerns how a session initiator finds an ingress member and authenticates itself to the overlay as a source of legimate packets. SOS is intended for use during an emergency situation, when networks are so congested that even benign ordinary traffic must be dropped. The members of SOS are hosted by a peer group of machines cooperating to provide emergency services. The only allowed packets to a given destination come from a few pre-configured sources used by emergency responders. So in SOS the source is itself an overlay member, *i.e.,* it has special software. SOS source members know the Internet names of many ingress members, well-distributed so that they cannot all be overwhelmed by flooding attacks. It creates a secure link to an ingress member, using ESP with endpoint authentication.

Mayday emphasizes an authentication option that is architecturally more complex, but has broader applicability because the source need not be an overlay member (both Figures 23 and 24 depict this option). In this option packets from a source to target name T are routed to some machine with an ingress member of the overlay. This can be accomplished by IP anycast, which will route packets from any source to the destination with name T closest to them. The underlay IP interface accepts the TCP session and passes control to the ingress member, which can then authenticate the source by asking for a user name and password associated with the target service. If the source is authentic, the ingress member initiates a TCP session *through the overlay* to the target; these two TCP sessions then become two parts of a compound application session.

Note from Figure 24 that the target server receives as source name the underlay name of the initiator. This means that reverse packets, from the Web server to the initiator, do not travel through the overlay. Although the path of return packets is not shown in the figure, both SOS and Mayday work like this.

Lightweight authenticator

In addition to the overlay network, a potential target must be surrounded in the Internet underlay by a ring of ordinary packet filters. These ordinary filters, such as firewalls or filtering forwarders, must have the capacity to handle flooding DoS attacks, and must be configurable by overlay machines or by people

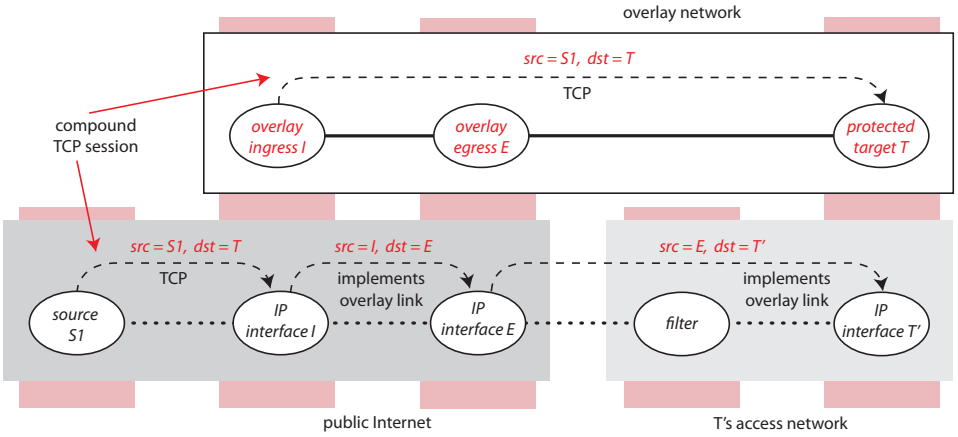

Figure 24: A session with overlay-based positive filtering, illustrating the following options: source is not an overlay member, target has different names in overlay and underlay, routing is singly-indirect.

representing the overlay.

Figure 24 is a session view of allowed access to protected target T. The last overlay hop between an egress member of the overlay and the target is implemented by an underlay path that goes through a filter. The lightweight authenticator is the attribute of underlay packets from an egress member to the target that causes the packet filter to recognize them as overlay packets and allow them to pass. The simplest lightweight authenticator is the IP name of the egress member (here E) in the source field of a packet; this is what SOS uses. Other authenticators proposed by Mayday include the destination port, destination name, and other header fields whose contents can be manipulated by the egress member.

The critical property of a lightweight authenticator is that it must be a secret—if attackers knew it, they could simply send underlay packets that match it. You might think that the destination name is the worst possible authenticator, but it can be a good one if the underlay name of the protected target is different from its overlay name, as shown in Figure 24, and if it can be changed easily and frequently by local control in the target's access network.

Overlay routing

In addition to hiding whole packet paths, overlay routing keeps the identities of egress nodes secret, which is indispensable if the lightweight authenticator is the name of an egress node. SOS uses egress names as authenticators; this is safe because of its elaborate overlay routing.

Mayday takes the position that effective overlay routing can be much simpler, with options including no routing at all (ingress and egress nodes are the

same), and singly-indirect routing (one hop between ingress and egress nodes, as in Figure 24). The Mayday paper reports on analysis showing that certain combinations of overlay routing and lightweight authenticator provide "best cases" for trade-offs among performance and security. For example, it says that designers who want moderate levels of both performance and security should use singly-indirect routing with any authenticator other than egress name.

6.2.4 Overlays for anonymity

In §6.1.2 we showed how joinbox/proxies in transit networks can provide session-initiating users with privacy within their access networks and anonymity at the 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 [14, 43] is to add to the services above a high degree of privacy from the proxies. This section describes the second, current Tor design [14].

Tor is an overlay network whose infrastructure members reside on the machines of volunteers world-wide. These infrastructure members are fully connected by long-lived links, each of which is implemented by a TLS session in the public Internet. This covers two of the ways Tor makes use of 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 (along with a certificate) to authenticate itself when setting up links by means of TLS sessions.[2]

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 25. 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 25, 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}.

[2] In Tor terminology, a proxy is an "onion router" and a user member is an "onion proxy." But Tor proxies do no routing, and user members do no proxying.

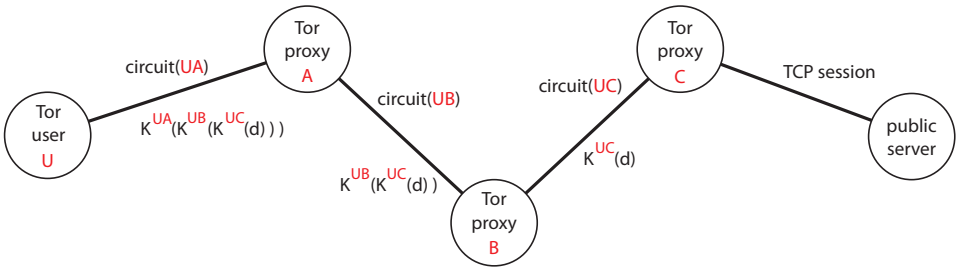

Figure 25: A compound session made by Tor. The first three simple sessions use the Tor circuit protocol, and go through the Tor network. The last simple session uses TCP, and goes through the public Internet.

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

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 inside TCP. Tor has a second session protocol, the stream protocol, which is embedded in the circuit protocol. Figure 26 shows all the session protocols, with protocols above embedded in protocols below, used for a single TLS application session through Tor.

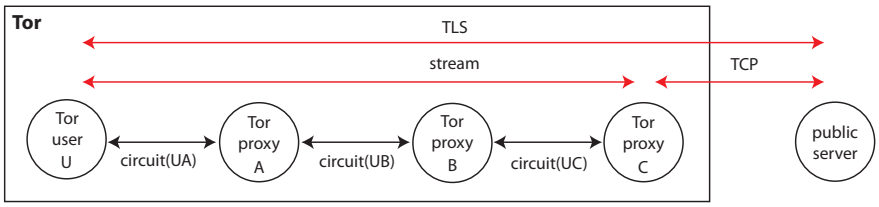

Figure 26: Session protocols and their embeddings, for a single TLS application session made through Tor. Sessions of the circuit protocol last longer than application sessions.

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.

Tor is a complex network in its own right. We have seen that it has its own session protocols, compound sessions, and routing. It also has internal mechanisms for denial-of-service protection and congestion control. Rate-limiting can be managed on circuits or on streams.

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 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 [29]. 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.

Although the Tor-exit problem appears to have no solution in today's Internet, privacy is so valuable that some researchers propose to replace much of IP with the concepts of Tor [34]. If the use of Tor became much more widespread as this proposal argues it should, then the stigma of using Tor would fade away.

7 Conclusion

From immersion in the literature on network security, one gets a strong impression. Each paper responds to a specific known or suspected security threat with a specific defense—which is often complex. It is hard to believe that network operators and users will ever be able to deploy all of these defenses, and troubling to think that they will have to choose some arbitrary subset of them. Most of these security papers are excellent, and they use careful and subtle reasoning to enumerate possible attacks and discuss which ones their defenses should hold against. But there are so many that one is driven to think, "What an insightful list! But how do I know that they thought of *everything*?"

This is another way of saying that modeling and assurance of 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 we have used a new, compositional model, which emphasizes that there are many networks in an overall network architecture, each one being a microcosm of all the basic aspects of networking. Because all networks are similar in important ways, they can be modeled with common structures and common compositional interfaces. This modularity greatly expands the range of network behaviors that can be modeled without overwhelming complexity [51].

In this tutorial, the new model has enabled us to find a useful classification of security attacks, and to explain all common defenses by means of just three patterns. It has also helped us understand how the patterns interact, without limiting where or how the patterns are used in an overall network architecture.

Not surprisingly, the modeling and defenses are not complete. The most obvious gap concerns information leakage (§2.3), in which attackers correlate observations from multiple locations, or from multiple explicit and implicit signals, to glean private information.

Nevertheless, this is a big step forward in modeling networks and their security mechanisms, which we hope will stimulate progress in building and verifying real defenses. It has been shown that real software can be made secure with modularity and verified components [31]. Technological progress has made network hardware increasingly programmable, so that flexibility and evolution are achievable. The synergy of improved modeling, programmable networks, and today's verification tools could provide the world with notably better network security.

References

[1] D. G. Andersen. Mayday: Distributed filtering for Internet services. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*, 2008.

[3] K. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *Proceedings of the USENIX Annual Technical Conference*, 2005.

[4] K. Argyraki and D. R. Cheriton. Network capabilities: The good, the bad, and the ugly. In *Proceedings of the 4th Workshop on Hot Topics in Networks*, 2005.

[5] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski. The SCION Internet architecture. *Communications of the ACM*, 60(6):56–65, June 2017.

[6] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *Proceedings of ACM SIGCOMM*, 2017.

[7] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of ACM SIGCOMM*, 2016.

[8] M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[9] R. Canetti. Universally Composable Security: A new paradigm for cryptographic protocols. https://eprint.iacr.org/2000/067.pdf, 2018. Accessed 22 January 2019.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proceedings of SIGCOMM*. ACM, August 2007.

[11] D. D. Clark. The design philosophy of the DARPA Internet protocols. In *Proceedings of SIGCOMM*. ACM, August 1988.

[12] D. D. Clark. *Designing an Internet*. Information Policy Series, MIT Press, 2018.

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

[14] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[15] Dyn analysis summary of Friday October 21 attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/. Accessed 10 November 2018.

[16] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A general approach to network configuration analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015.

[17] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, 2012.

[18] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of Symposium on Theory of Computing*. ACM, 2013.

[19] T. Grandison and M. Sloman. A survey of trust in Internet applications. *IEEE Communications Surveys and Tutorials*, 3(4):2–16, 2000.

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

[21] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*, 2001.

[22] How does TLS affect MQTT performance? https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/. Accessed 19 September 2018.

[23] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama. TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency. In *Proceedings of SPIE*, volume 6011, pages 138–146. International Society for Optical Engineering, 2005.

[24] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2011.

[25] ITU. Information Technology—Open Systems Interconnection—Basic Reference Model: The basic model. ITU-T Recommendation X.200, 1994.

[26] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable Internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*. USENIX, 2011.

[27] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using Header Space Analysis. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013.

[28] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of SIGCOMM*. ACM, August 2002.

[29] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, V. Paxson, S. J. Murdoch, and D. McCoy. Do you see what I see? Differential treatment of anonymous users. In *Proceedings of the Network and Distributed Security Symposium*. Internet Society, 2016.

[30] T. Kiravuo, M. Sarela, and J. Manner. A survey of Ethernet LAN security. *IEEE Communications Surveys & Tutorials*, 15(3):1477–1491, 2013.

[31] G. Klein, J. Andronick, M. Fernandez, I. Kuz, T. Murray, and G. Heiser. Formally verified software in the real world. *Communications of the ACM*, 61(10):68–77, October 2018.

[32] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach, Seventh Edition.* Pearson Education, Inc., 2017.

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

[34] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson. Tor instead of IP. In *Proceedings of the 11th Workshop on Hot Topics in Networks*, 2011.

[35] N. P. Lopes, N. Bjorner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015.

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

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

[38] G. C. M. Moura, J. Heidemann, M. Muller, R. de O. Schmidt, and M. Davids. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the ACM Internet Measurement Conference*, 2018.

[39] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafo, K. Papagiannaki, and P. Steenkiste. The cost of the 'S' in HTTPS. In *Proceedings of ACM CoNEXT*, 2014.

[40] D. Naylor, R. Li, C. Gkantsidis, , T. Karagiannis, and P. Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of ACM CoNEXT*, 2017.

[41] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *Proceedings of ACM SIGCOMM*, 2015.

[42] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable Multipath TCP. In *Networked Systems Design and Implementation*, 2012.

[43] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[44] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of SIGCOMM*. ACM, 2000.

[45] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster. Oblivious DNS: Practical privacy for DNS queries. arXiv:1806.00276v2, December 2018.

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

[47] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep packet inspection over encrypted traffic. In *Proceedings of SIGCOMM*. ACM, 2015.

[48] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

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

[50] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *Proceedings of SIGCOMM*. ACM, August 2005.

[51] P. Zave and J. Rexford. The compositional architecture of the Internet. *Communications of the ACM*, 62(3), March 2019.

[52] L. Zhang, A. Afanasyev, J. Burke, and V. Jacobson. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, July 2014.